Aplicación móvil para un proyecto IoT

Proyecto de fin de carrera

Abstract

[Estudio y desarrollo de una aplicación nativa Android para visualizar datos de proyectos IoT. Se centrará en la visualización de distintos datos, a los que se accederá tanto de manera externa como interna del dispositivo móvil]

Miriam Calatrava Sierra

Tutor: Miguel Ángel Patricio Guisado
Director: Álvaro Luis Bustamante
Universidad Carlos III – Ingeniería en informática

Agradecimientos

La finalización de este trabajo y por lo tanto de esta etapa de la vida no hubiera sido posible sin el apoyo incondicional de mucha gente.

Por un lado a mis amigos de la universidad, que han sido totalmente comprensivos y han entendido que esta era mi prioridad. Se nota que habéis pasado por lo mismo.

Por supuesto al director del proyecto, Álvaro, que me ha guiado en todo el proceso y ha estado siempre dispuesto a echarme una mano y a dar respuesta a mis incesantes preguntas, y a Miguel Ángel, que sin su ayuda no habríamos podido llegar a este momento.

A mis tíos, primos y el resto de la familia que me preguntabais si ya lo había entregado, deciros que parecía que no iba a llegar nunca este momento, pero que finalmente ha llegado.

A mis amigas de toda la vida, Adela, Almu y Raquel que siempre habéis estado ahí cuando os he necesitado, incluso cuando no he querido necesitaros.

Por último, me gustaría agradecérselo sobre todo a mi familia, a mi abuela que de pequeña siempre me decía que estudiara, a mi hermana Silvia y a mi madre Consuelo que me han apoyado, motivado y animado para que no abandonara y aguantado que les explicara durante horas mis avances sin entender nada. En especial me gustaría dedicárselo a mi padre, José Luis, que lo único que ha querido en la vida es saber que he terminado con éxito mis estudios y que me ha ido mejor que a él en la vida, y aunque no vaya a poder estar en mi graduación, estoy segura de que se quedará más tranquilo al saber que por fin seré oficialmente una ingeniera. Esto lo hago por ti, para que te sientas orgulloso, aunque en mi caso no vayas a poder contarlo a tus conocidos, que sé que hablar bien de tus hijas era algo que te llenaba de felicidad.

Contents

Índice de imágenes

Índice de tablas

1. Introducción

1.1. Motivación

El motivo principal por el que se decidió realizar este proyecto era porque además de poder realizar la parte práctica y poder aprender algo nuevo relativo a la programación, era importante poder hacer algo novedoso o que se encontrase actualmente en auge, ya que no era muy apetecible desarrollar un proyecto de fin de carrera que no fuera a ser interesante o servirme para el futuro.

A día de hoy, es muy importante recalcar la importancia que están adquiriendo los proyectos IoT (Internet of Things) y esta tendencia se encuentra actualmente en ascenso. Esto se debe a las múltiples aplicaciones que tiene en el mundo real [5]:

- Sistemas de tráfico inteligente → se podrían descongestionar zonas con alta presencia de vehículos. Se monitorizarían las carreteras y se reportarían los accidentes en el mismo momento en el que se produjesen, al igual que cualquier otro cambio que afecte al tráfico.
- Medioambientales → se utilizaría para la predicción de desastres naturales.
- Casas inteligentes → con esta tecnología se podrían ahorrar recursos de agua y energía teniendo en cuenta los datos obtenidos del exterior como humedad en el ambiente, luminosidad, temperatura, etc.
- Hospitales inteligentes → capaces de monitorizar a los pacientes con pulseras que contendrán todos los datos médicos y podrán dar información en tiempo real de la presión arterial, ritmo cardiaco, etc. De hecho, se ha comenzado a hablar de la utilización de drones como ambulancias de atención temprana que puedan transportar kits de emergencia para ayudar al herido hasta que lleguen los profesionales.
- Agricultura inteligente → gestionarían el riego automático de las cosechas según datos de temperatura, humedad en el ambiente y luminosidad para maximizar la producción.
- Tiendas inteligentes → se podría controlar y monitorizar los cambios en el stock disponible e incluso detectar robos, de tal manera que se realicen los pedidos de forma automática.

Además, no sólo se están utilizando en ámbitos de investigación o amateur por personas interesadas en estos sistemas, sino que llevan un tiempo siendo desarrollados en el ámbito comercial y empresarial.

En el ámbito empresarial es posible encontrar dispositivos IoT como por ejemplo las pulseras de muñeca para los deportistas que monitorizan sus pulsaciones, horas de sueño, tanto ligero como profundo, etc. O los relojes llamados 'inteligentes' que envían datos sobre localización, datos de rutas de transporte y consultas de tráfico, etc. Hay múltiples productos que cuentan con sus propias aplicaciones personalizadas al dispositivo que se utiliza. Otro ejemplo de este estilo podría ser la iniciativa de Google Car, que permitiría la experiencia de la conducción con tráfico en tiempo real, con condiciones atmosféricas y de carretera variable.

Dentro de estos sistemas y de sus múltiples aplicaciones, se observó que actualmente no hay grandes alternativas a la hora de poder monitorizar los datos recibidos de proyectos IoT propios de manera visual con una aplicación nativa. Sí que existen diversas plataformas para estos desempeños, pero habitualmente son de escritorio, ya que la oferta nativa para dispositivos móviles es más reducida. No sólo se quería que los datos pudieran ser visualizados de forma gráfica, sino que se quería dar la opción de personalizar el modo de verlos y ofrecer además otras funcionalidades que aportaran información extra relevante sobre el proyecto.

Debido a la enorme expansión de estos sistemas y viendo las múltiples aplicaciones y posibilidades que ofrece en multitud de sectores se decidió realizar una aplicación móvil nativa de monitorización de datos. Por lo tanto, el objetivo principal del proyecto es ofrecer a los usuarios una aplicación móvil desde la que sean capaces de consultar todos los datos de su proyecto IoT y estar al tanto de las últimas actualizaciones del estado del mismo.

1.2. Objetivos

El objetivo principal que se quiere conseguir con este proyecto es el del desarrollo de una aplicación móvil que se centra y sitúa dentro del marco del Internet of Things para poder monitorizar gráfica y visualmente ciertos valores concretos que se obtienen de dispositivos, tanto del propio terminal móvil como de datos obtenidos de forma remota a través de la red.

En resumidas cuentas, se quiere desarrollar una aplicación nativa móvil, concretamente Android por el número de usuarios que lo usan actualmente, que muestre gráficamente datos. Dichos datos, dependiendo de su naturaleza se podrán obtener de manera de externa, con conexiones a la red, o de dentro del dispositivo, para lo cual se debe poder gestionar los accesos al mismo. La representación de los mismos tendrá distintas variantes según el tipo de dato.

Además, debe ser posible mantener información entre sesiones, lo que quiere decir que se deben persistir tanto los datos como su representación, y debe poderse estructurar la información de algún modo lógico que permita al usuario organizar la información de la manera deseada.

1.3. Esquema del documento

El documento de memoria del proyecto cuenta con una serie de apartados y es importante enumerarlos para entender mejor la estructura del mismo. Los apartados con los que cuenta, sin contar con este introductorio, son los siguientes:

Estado del arte → en este apartado se expondrá toda la investigación previa que se ha tenido que llevar a cabo antes de empezar el desarrollo. Se comenzará explicando otras aplicaciones que ofrecen funcionalidades similares a lo que se pretende conseguir para establecer un punto de partida y unos objetivos claros de cara al proyecto. A continuación, se documentará todo el proceso de la elección de una librería de gráficos para Android, añadiendo pruebas gráficas y definiendo las ventajas e inconvenientes de cada una de ellas.

Diseño & desarrollo → en el que se enumerarán los pasos y fases del desarrollo de la aplicación, empezando por la definición de una estructura básica del proyecto y el uso de sus directorios. Una vez hecho eso, se explicarán todas y cada una de las secciones tanto en términos de procesamiento de datos como comunicaciones externas, integración de la librería de datos escogida y visualización de las pantallas.

Medios empleados → determina los medios que se han utilizado en términos de hardware y software para la finalización de la aplicación. En esta sección se tratarán los terminales móviles, además de algunos periféricos, y la elección de Android Studio como IDE (entorno de desarrollo integrado).

Planificación → estima los tiempos necesarios para llevar a cabo el proyecto y las herramientas de monitorización que se utilizan para controlar las desviaciones lógicas que se producen en cualquier proyecto de una cierta envergadura.

Presupuesto → divide las partidas económicas en términos de personal, software y hardware para controlar los costes y presupuestos del proyecto.

Conclusiones y próximos pasos → este apartado es totalmente clave, ya que no sólo resumirá todo lo que haya realizado para este proyecto y cuál ha sido el resultado final teniendo todo eso en cuenta, sino que enumerará nuevas líneas de desarrollo que podrían ser añadidas posteriormente, pero que no han sido añadidas a este proyecto porque se escapaban de nuestros objetivos. Es decir, que define tanto el estado actual del proyecto como un posible estado futuro con funcionalidades y mejoras extra.

Referencias → contiene las referencias a artículos, libros y documentación que ha sido consultada y estudiada para la realización de este proyecto y por supuesto, para la redacción de este documento.

2. Estado del arte

En este apartado lo que se tratará de hacer, es explicar y exponer todos aquellos elementos que se encuentren relacionados con el desarrollo de la aplicación. Se comenzará con el concepto de IoT, ya que es en este tipo de tecnología en la que basará el desarrollo. A continuación, se estudiarán algunas plataformas ya existentes que sirven para gestionar proyectos IoT y, por último, se centrará en la exposición de los distintos tipos de gráficas de representación de datos y de librerías en Android que los implementan.

2.1. Internet of Things

IoT son las siglas de “Internet of things” o como se conoce comúnmente en español “el internet de las cosas”. Este término recoge una idea que consiste en la interconexión de dispositivos físicos, ya sean o no de uso cotidiano, a internet.

El IoT ofrece la oportunidad de integrar directamente objetos físicos dentro de sistemas computerizados, de tal manera que esto resulte en una mejora beneficiosa en términos de eficiencia y precisión reduciendo al máximo la intervención humana, ya que el concepto comprende que cualquier objeto pueda interconectarse y comunicarse con cualquier otro de forma remota, haciéndolos más independientes de las personas. Estos artefactos deberían ser capaces de recabar datos sobre el entorno en el que se encuentran a través de sensores, cámaras, etc. procesar estos datos y realizar acciones según los resultados. Es posible que los resultados del procesamiento requieran la comunicación de estos datos a otros dispositivos para acciones futuras por lo que deberán estar interconectados entre sí por algún tipo de red.

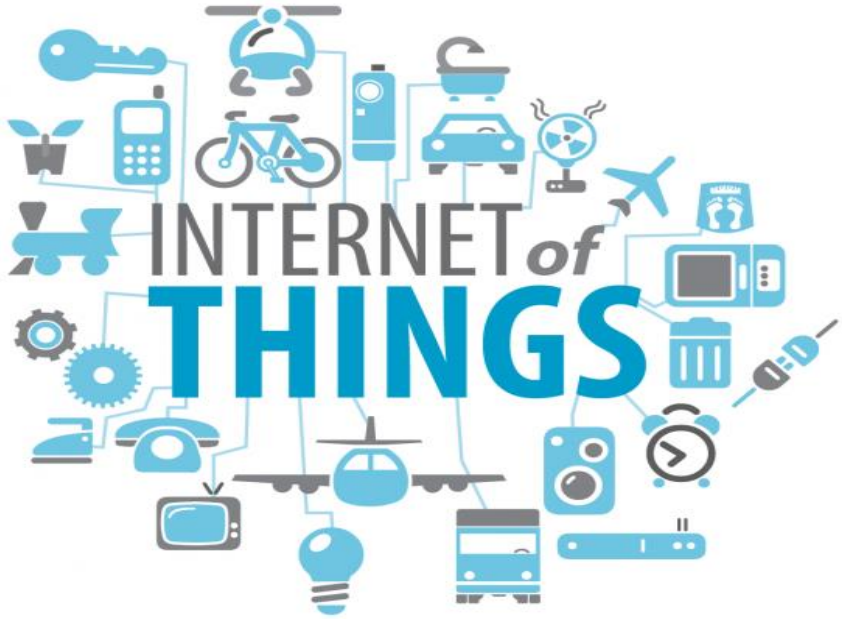

Figure 1: Internet of Things

Este concepto que data al menos de 1985 y fue descrito por Peter T. Lewis en una conferencia para la fundación Congressional Black Caucus [27], engloba también otras tecnologías igualmente populares dentro de la industria informática, como puede ser los hogares inteligentes, las ciudades inteligentes o el transporte inteligente. Para aclarar a qué se refiere el documento exactamente con lo de 'inteligentes' y de qué tipo de tecnologías se está hablando es posible utilizar el siguiente ejemplo. Imaginemos que nuestra nevera revisara periódicamente los alimentos que actualmente nos faltan en la misma y que utilizando el conocimiento previo de cuáles son nuestros hábitos alimenticios hiciera

una compra online para no quedarnos nunca desabastecidos o que la calefacción se activase en el momento concreto en que nos dirigimos a casa porque ha recibido nuestra ubicación actual y la ruta que estamos siguiendo es la que utilizamos habitualmente cuando nuestro destino es nuestra casa, etc. Es decir, se les denomina 'inteligentes' porque actuarán de manera independiente según datos externos que no serán proporcionados directamente y a propósito por el usuario, sino que los recogerán por su cuenta. La tecnología incorporará otros campos de la informática como la comunicación remota, análisis y procesamiento de datos en tiempo real, machine learning o aprendizaje automático y distintos componentes hardware electrónicos como, por ejemplo, sensores.

Con el tiempo, el concepto original ha ido cambiando y adoptando nuevas características que incluso no tienen por qué ser las definitivas, pero actualmente se define como una red de objetos interconectados entre sí, que se organizan y gestionan de manera autónoma e inteligente basándose en el contexto o entorno en el que se encuentren. Según esta definición se tendrán que tener en cuenta entre otras las siguientes consideraciones [3, 9]:

- Tamaño → será absolutamente necesario tener en cuenta que el número de dispositivos conectados aumentará y que el sistema deberá estar preparado para controlar y monitorizar grandes cantidades de aparatos y de los mensajes que se envíen entre ellos [5].
- Espacio → al hablar de artefactos físicos se debe ser consciente del espacio físico real que ocupan cada uno de ellos y de dónde situarlos geográficamente, ya que existirán algunos que dependan totalmente de su localización como por ejemplo sensores de luminosidad.
- Tiempo → al tener tantos objetos conectados existirán múltiples mensajes enviados a la vez, es decir, simultáneos, por lo que se deberá controlar el procesamiento de los mismo de forma paralela y en tiempo real.
- Seguridad → según algunos estudios [2], se ha llegado a confirmar que un alto porcentaje de dispositivos IoT tienen problemas de seguridad y vulnerabilidades en el cifrado de datos enviados y en el acceso a los mismos. En la actualidad se han llegado a realizar diversos experimentos en los que se hackeaban los sistemas de los dispositivos [23], tanto para el robo como para la modificación de datos.
- Protección de datos y privacidad → no sólo hay que tener en cuenta que los datos pueden ser accedidos fácilmente por fallos de seguridad, sino que muchos de estos dispositivos obtendrán datos directamente del ambiente y no se respetará la privacidad de aquellos humanos que no quieran formar parte de esta red. Es decir, si quieren formar parte de la sociedad tendrán que rendirse a estar bajo supervisión continua. Es posible que en el futuro se idee algún sistema para solventar esta problemática, pero de momento no se ha definido nada al respecto, aunque sí que se han llegado a hacer las siguientes recomendaciones por la Comisión federal de comercio (FTC) [28]:
  - Seguridad de los datos → todo aquel dato que se recoja, se procese y se almacene, debe pasar por un proceso seguro en todas las fases.
  - Consentimiento de los datos → los usuarios de los que se vayan a recabar datos deben dar su consentimiento explícito a la extracción de los mismos.
  - Simplificación de la recogida de datos → define que los dispositivos sólo deben recoger y almacenar aquellos datos que realmente sean necesarios y siempre de manera temporal.

La cuestión es que, aunque estas recomendaciones sean algo de sentido común, no se ha empezado siquiera a plantear cómo podrían llevarse a cabo en el mundo real.

En base a estos puntos, en un futuro sistema global de IoT se debería contar con las siguientes características [5, 9]:

- Arquitectura → deberá ser una arquitectura orientada a eventos [10]. De tal manera que los cambios producidos en el entorno originen modificaciones o acciones en tiempo real.
- Arquitectura de red → será un requisito indispensable que la red sea escalable, ya que se estima que los objetos sujetos a IoT serán numerosos y que no harán más que aumentar esta cantidad con el tiempo. Para poder contar con esta característica el desarrollo de Ipv6 tendrá una gran importancia, ya que se usará para poder identificar los dispositivos.
- Red → se necesitarán tecnologías que permitan el envío de mensajes entre los dispositivos. Para ello, sería posible dividir las conexiones según la distancia entre dispositivos y utilizar distintas tecnologías de acuerdo con las necesidades de cada uno de los bloques divisorios.
- Complejidad → debido al gran número de interconexiones e interacciones posibles se considera que el entorno será potencialmente caótico, pero se pretende mitigar este efecto desarrollando subsistemas en los cuales los dispositivos sólo se comuniquen internamente y únicamente funcionen dentro del mismo.

El impacto que tendrá esta tecnología en un futuro cercano será inconmensurable tanto en términos económicos como en cómo se relaciona el ser humano con el mundo que le rodea y por lo tanto afectará a la sociedad en su conjunto, ya que abre un gran abanico de posibilidades. De hecho, según algunas estimaciones se calcula que para 2020 existirán en el mundo más de 25 mil millones de objetos utilizando esta tecnología [5, 28], así que parece que el IoT se integrará en nuestras vidas a medio y largo plazo.

Algunas de las aplicaciones que se prevén para el futuro ya han sido expuestas en la introducción, pero es importante enumerarlas también en este mismo apartado para recogerlas en un único conjunto [29]:

- Casas inteligentes → las tecnologías IoT podrían usarse para controlar el uso de la energía, programar y planificar la calefacción o el aire acondicionado teniendo en cuenta las horas de llegada del usuario, apagar las luces automáticamente cuando no haya nadie en una habitación, desbloquear la puerta principal para algunas personas automáticamente o incluso el frigorífico podría realizar automáticamente la compra según las necesidades y costumbres del usuario
- Dispositivos wearable → actualmente se están potenciando estos dispositivos que se encargan de recoger datos personales del usuario como puede ser información sanitaria con las constantes vitales o de gustos e intereses
- Industrial IoT → este sector se centraría en la monitorización del inventario, la creación de pedidos automáticos cuando se vayan agotando las existencias
- Ciudades inteligentes → control de los recursos tanto físicos como energéticos en tiempo real según las características medioambientales, monitorización del tráfico, búsqueda de plazas de aparcamiento libres, gestión del sistema de semáforos y señalización

2.2. Plataformas de gestión de proyectos IoT

En este apartado se estudiaran una serie de plataformas y aplicaciones móviles de visualización de datos y gestión de proyectos IoT que también se utilizarán como base para el desarrollo y estructuración del proyecto. No sólo se estudiarán plataformas nativas, sino que se expondrán otras alternativas para dispositivos móviles.

2.2.1. Blynk

Blynk [11] es una plataforma para administrar distintos dashboards o tableros compuestos por drag&drop widgets que se comunican con dispositivos Raspberry PI, Arduino, etc, para conseguir sus datos. En otras palabras, es una aplicación que sirve para crear proyectos IoT propios.

Al abrir la aplicación, se le solicita al usuario crear una cuenta asociada a través de una dirección de correo electrónico. Esta cuenta será la que se utilizará a la hora de crear los distintos proyectos, de tal manera que todos se encuentren relacionados entre sí y puedan ser administrados desde el mismo dispositivo. Esto posibilita tener múltiples tableros, es decir, proyectos, aunque todos ellos comparten una limitación de la que se hablará a continuación.

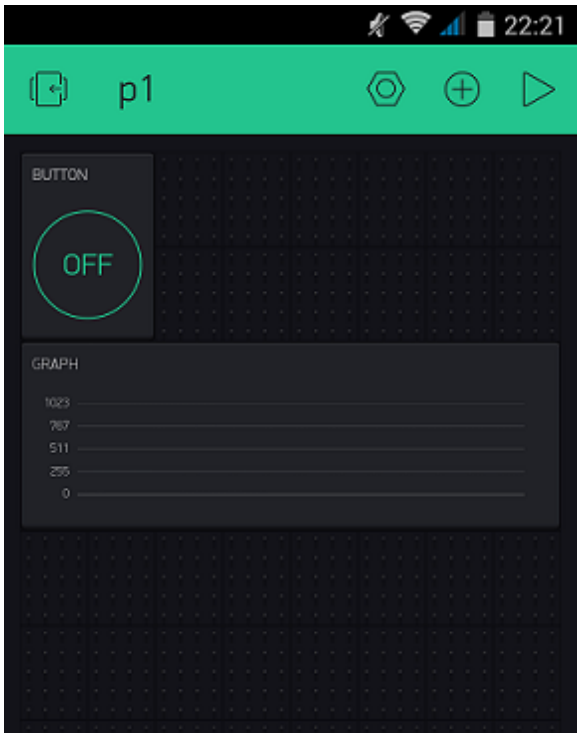

Figure 2: Blynk Dashboard

La aplicación proporciona a los usuarios un número fijo de puntos, llamados energía, que se van consumiendo según se utilizan widgets en el tablero. Según el tipo de widget la cantidad de puntos que se han de canjear varía. La cantidad asignada a cada uno de ellos está relacionada a la complejidad del mismo widget o a la funcionalidad que representa. Los puntos son además compartidos por todos los proyectos del usuario, lo que limita enormemente el uso que se puede dar a la herramienta.

Los widgets se dividen en varias categorías:

- Controladores → contiene botones, sliders, timers, etc. que permiten la activación/desactivación y control de nuestro dispositivo
- Displays → ofrece la posibilidad de mostrar los resultados y valores obtenidos de nuestros dispositivos de forma gráfica
- Interfaz → permite ordenar más fácilmente la información en el tablero
- Notificaciones → Añade la posibilidad de notificar por diferentes medios (Twitter, email) información relativa al tablero
- Sensores del teléfono → incluye sensores propios del teléfono como por ejemplo el sensor de luz o el GPS
- Otros → contiene otros widgets que no encajan en el resto de categorías como por ejemplo un reloj de tiempo real

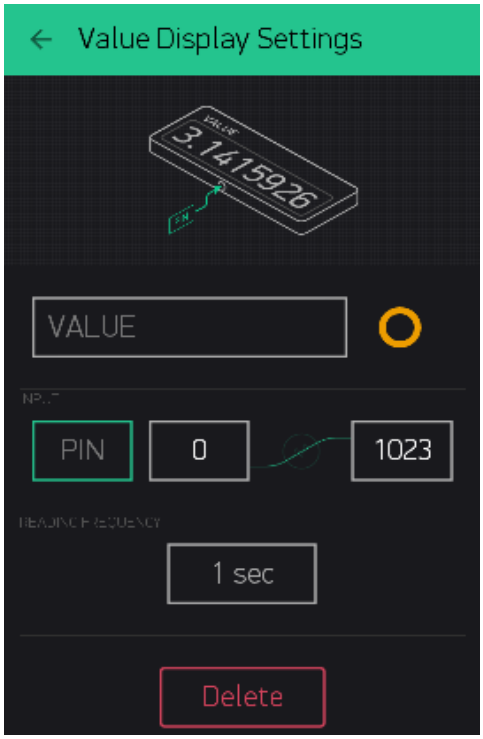

Figure 3: Configuración de widget en Blynk

Por último, existe la opción de compartir un proyecto con otros usuarios y no restringirlo a nuestro dispositivo (móvil). Esta opción no permite que el resto de usuarios modifique el proyecto, pero sí que se les actualiza la versión con los últimos cambios y también pueden controlar los widgets añadidos. El problema de esta funcionalidad es que es canjeable por la mitad de los puntos que se tiene en un inicio, y que en caso de quitarla no devuelven los puntos por lo que es mucho menos flexible que la distribución de los widgets, la cual es posible intercambiar al gusto tantas veces como sea necesario.

Además de tener en cuenta los puntos también debe el usuario fijarse en el espacio físico de tablero, ya que los tableros están limitados a 72 cuadrados y el tamaño de los widgets son variables.

En cuanto al software de la aplicación de Blynk para dispositivos móviles, se puede comentar que es un software propietario. Por el contrario, la librería que el propio equipo de desarrolladores ha creado para que sea instalada en el dispositivo hardware es software libre y puede encontrarse y estudiarse en github al igual que el servidor, que es utilizado para la transmisión de mensajes entre la aplicación móvil y el propio dispositivo. Ambas librerías están desarrolladas en C++.

Actualmente los creadores están trabajando principalmente es dar soporte a un número mayor de dispositivos hardware, pero deberían trabajar no sólo en esta rama del conocimiento, sino que deberían permitir un mayor control de los componentes físicos que acompañan al dispositivo. Una funcionalidad que gestione los eventos que sucedan durante la ejecución del proyecto sería una gran ayuda y flexibilizaría el uso del dispositivo, utilizando mejor el potencial que realmente tiene y que se está desaprovechando.

2.2.2. Cayenne

Cayenne [12] es una aplicación nativa para smartphones IOs y Android que permite crear proyectos IoT para desarrolladores que utilicen Raspberry Pi o Arduino, este último se encuentra actualmente en fase beta.

Antes de empezar a generar el proyecto, el usuario debe conectar el dispositivo tanto a la red como a la propia aplicación y partir de ese momento ya es posible crear un tablero y empezar a añadir controladores de sensores, componentes de alertas, etc.

La plataforma se compone de múltiples componentes:

- Cayenne App → La aplicación que se encarga de crear los paneles con widgets para los proyectos IoT
- Cayenne Online Dashboard → El panel de control online al que se puede acceder desde un navegador y que sirve para monitorizar el proyecto. Tanto la App como el Online Dashboard se usan para lo mismo y es decisión del desarrollador si quiere usar uno, el otro o ambos según conveniencia, ya que los cambios se reflejan en los dos componentes
- Cayenne Cloud → Almacena datos sobre los dispositivos, usuarios y sensores de los que se compone el proyecto
- Cayenne Agent → gestiona todo tipo de comunicación que se produzca entre el hardware del dispositivo, la aplicación y el servidor de datos para permitir alertas y acciones reactivas

Figure 4: Cayenne Dashboard

El proceso de comunicación según explicado por los propios desarrolladores es el siguiente:

Una vez que se produce cualquier modificación sobre el proyecto IoT definido, ya sea desde la aplicación móvil o desde el Online Dashboard, se envía el dato al Cloud donde se procesa y se envía al hardware. De la misma manera ocurre cuando el mensaje surge del propio hardware y llega hasta la aplicación.

La aplicación es capaz de integrar nuevos componentes como actuadores (relés, motores, etc.) o sensores (de movimiento, de presión, etc.) y muestra al usuario el listado de puertos de acceso, conocidos como GPIO (General Purpose Input/Output, Entrada/Salida de Propósito General), para el dispositivo. De esta manera, el usuario puede controlar completamente el hardware desde la aplicación.

Una vez integrados los componentes hardware en el panel (habitualmente a través de widgets que controlen su funcionamiento o que muestren los datos que recaban esos componentes, ya que cada sensor/actuador tiene asociado al menos un widget), se pueden crear condicionales o alertas que activen ciertas acciones en el proyecto IoT, como puede ser enviar una notificación email de que se ha superado un valor en un sensor concreto o encender algún piloto luminoso. Además de crear

triggers que activen acciones específicas en el proyecto, se pueden planificar cambios en el mismo a través de un calendario.

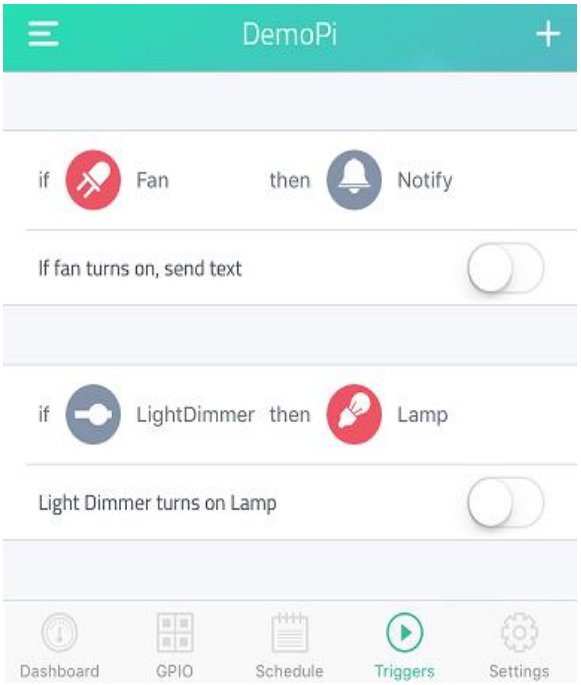

Figure 5: Configuración de widget en Cayenne

La aplicación permite introducir software personalizado a través de ficheros sketches (.INO) para Arduino, y para ello proporciona desde github una librería que contiene ejemplos de cómo deben estructurarse estos sketches. La posibilidad de crear nuestros propios widgets proporciona gran flexibilidad a la herramienta, ya que es posible tener control total sobre cómo manda el Arduino los datos a la aplicación de Cayenne y cómo ésta los procesa. Desde github se puede observar que la librería de Cayenne contiene a su vez código de la librería de Blynk, pero una versión que está actualmente desfasada, por lo que esta opción pierde bastante fuerza, aunque siempre existe la posibilidad de que decidan actualizar la versión en un futuro.

En resumidas cuentas, Cayenne es una aplicación para crear proyectos IoT bastante flexible y con un gran potencial que podría mejorar enormemente si diera soporte a más dispositivos, ya que actualmente sólo lo hace para Raspberry PI.

2.2.3. Thinger

Thinger [13] es una aplicación web desarrollada para crear proyectos IoT. La aplicación se encarga de crear la infraestructura necesaria para comunicar dispositivos IoT con un panel de control visual escalable compuesto de widgets, de tal manera que los usuarios sean capaces de conectar sus dispositivos y empezar a manejarlos en cuestión de pocos minutos. La plataforma está alojada en la nube mientras que el código es open source y puede encontrarse en GitHub.

La web cuenta con una versión 'Responsive Design' que permite su uso a través de un navegador en dispositivos móviles. Las tecnologías responsive permiten la adaptación de páginas web a las distintas resoluciones de pantalla, intentando con ello que se ajusten lo máximo posible y ofrezcan la mejor experiencia de usuario dentro de sus posibilidades y de la pericia del diseñador y del desarrollador, porque de ellos depende en gran medida que la adaptación sea lo más fiel posible a la original y que además su uso sea cómodo y accesible.

Figure 6: Thinger Dashboard

Thinger proporciona un listado de widgets que pueden clasificarse en dos categorías distintas; visualización y control.

- Muestreo de datos
  - Timeseries o línea temporal → Mostrará distintos valores y su evolución en el tiempo. Los principales puntos de este elemento son el número de muestras que se tomarán por unidad de tiempo y qué rango de tiempo se quiere mostrar. La unidad del tiempo puede ser variable y no tiene ni siquiera por qué medirse en tiempo, sino que puede recogerse este dato cada vez que sufre modificaciones (es decir, se actualizaría según un evento de cambio) o en un rango de tiempo concreto (es decir, se actualizaría de forma periódica independientemente de que el valor haya o no cambiado). También será posible decidir qué rango de tiempo se quiere mostrar, siempre teniendo en cuenta que el rango se calculará a partir del último valor del dato hacia atrás
  - Donutchart → Se usa para representar porcentajes o datos con una distribución porcentual. Entre todas las secciones que componen este elemento, se deberá completar el 100% cerrando el círculo por completo. Para ello será posible introducir un valor mínimo y máximo entre los que podrá oscilar nuestro dato
  - Progressbar → Sirve para representar el porcentaje de desarrollo, progreso, etc sobre un total del 100%. A diferencia del Donutchart, este widget representará un único valor en lugar de múltiples secciones. El Donutchart a su vez es capaz de representar la misma información que el Progressbar utilizando un dato y su negación, por lo que en este caso son completamente intercambiables, ya que este último también cuenta con valores mínimos y máximos configurables por el usuario, la única diferencia entre ambos es la visual
  - Google Maps → Se encarga de embeber un mapa de una localización concreta basándose en sus coordenadas de latitud y longitud. Estas coordenadas puede fijarse desde el panel de control o ser leídas del dispositivo hardware IoT con el que esté conectado
- Datos individuales
  - Reloj → Mostrará un reloj para poder controlar mejor la evolución de aquellos elementos que se actualizan periódicamente. El valor lo recogerá del propio reloj interno del ordenador
  - Contenido textual → Este widget ofrecerá un único valor numérico/textual, ya sea fijo o variable en el tiempo. La diferencia con otros widgets es que sólo representa un valor en

lugar de una evolución del mismo o una comparativa entre los distintos valores que puede tomar. En otras palabras, mostrará el último valor que tome un dato, es decir, el valor actualizado

- Contenido multimedia → Este widget representará un contenido multimedia, que en este caso se tratará de una imagen. La imagen podrá ser fija o recoger el dato de una cámara en el dispositivo móvil

- Otros widgets con los que contará la plataforma serán widgets de control, que se encargan de interactuar con nuestro hardware.
  - Switch On/off → Este elemento se encargará de activar o desactivar alguna funcionalidad/acción de nuestro dispositivo
  - Slider → Este elemento se encargará de enviar información numérica al dispositivo. Permite incrementar un valor desde un mínimo y hasta un máximo y podría usarse para calibrar por ejemplo la velocidad de un motor conectado a nuestro dispositivo

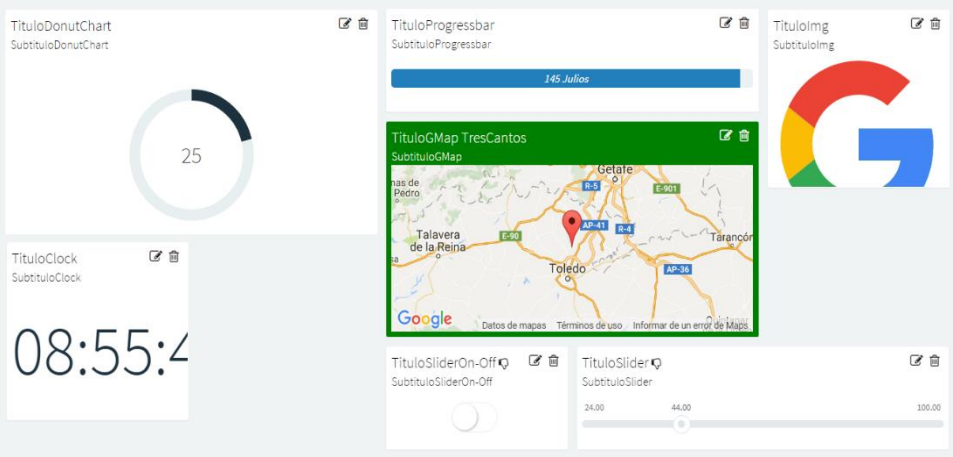

Figure 7: Selección de widgets de Thinger

Uno de los puntos fuertes y de interés de Thinger es que es una plataforma completamente escalable que permite no sólo la incorporación de múltiples paneles de control para gestionar los proyectos IoT, sino que para cada uno de ellos se pueden añadir distintos conjuntos de datos y widgets. En el caso de los widgets se puede personalizar tanto la posición de los mismos a través de un sistema clásico de drag and drop como el tamaño de los mismos o los colores con los que se representarán los valores. Eso es términos generales, pero como ya se ha visto se permiten otros tipos de personalizaciones dependiendo del componente.

Por otro lado, Thinger también cuenta con algunas carencias. Entre ellas se encuentra la falta de una aplicación móvil propia, aunque la plataforma web cuente con un diseño y desarrollo responsive, como ya se ha comentado al inicio de este apartado, que permite su uso en dispositivos móviles más o menos de manera adecuada, no es igual en términos de rendimiento, fluidez, etc. porque a final de cuentas lo que se realiza es una adaptación de la versión escritorio al dispositivo y siempre pierde en términos de rendimiento en una comparativa frente a una aplicación nativa. Otra carencia puede considerarse la variedad de widgets que permite introducir, ya que en comparación con alguna de las otras plataformas que tienen una trayectoria más larga, puede parecer que Thinger se quede corto con las opciones que ofrece, pero por otro lado y si se piensa bien, a priori tiene lo justo y necesario para el desempeño que va a hacer.

2.3. Visualización gráfica de datos

Un widget es un módulo que engloba una serie de funcionalidades concretas dentro de un paquete de software. Este tipo de módulos suele incorporar una parte gráfica que puede ofrecer o visualización de datos o sistemas de interacción con los mismos. Para este caso, un widget será un elemento que permita ver o interactuar a un usuario con los datos que se reciban de un dispositivo. Debido a que la manera en la que se representará la información será diversa, se tendrá que contar con distintos elementos gráficos.

2.3.1. Gráficas

Los gráficos son representaciones mediante figuras y signos de todo tipo de datos, de tal manera que el dato es transformado en un elemento visual determinado, con significado propio, como pueden ser barras o puntos en unas coordenadas.

Los tipos de gráficas más comunes son los siguientes:

- **Líneas temporales** para mostrar todo tipo de información que varía con el tiempo. Sirven principalmente para ver la evolución de un dato concreto con el paso del tiempo. Habitualmente estos gráficos muestran los datos en un rango de tiempo determinado por lo que su eje X suele estar marcado por unidades temporales.

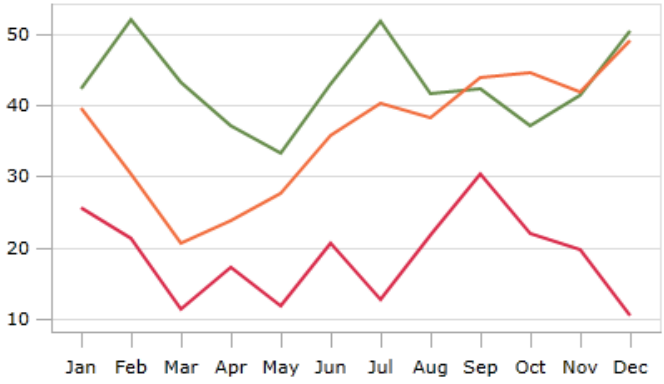

Figure 8: Gráfico de línea temporal

- **Gráficos de barras** que representan la frecuencia o el total de unos datos clasificados en diferentes categorías. De hecho, si se utilizase el tiempo como una categoría podrían ser comparables con las líneas temporales.

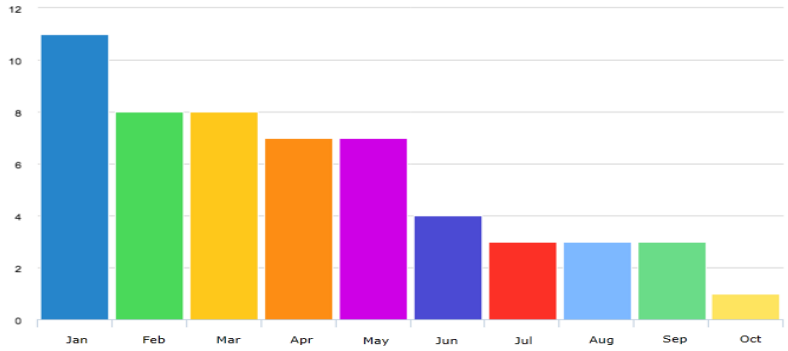

Figure 9: Gráfico de barras

- **Gráficos de tartas/donuts/piramidales** simbolizan los datos en forma de proporciones y porcentajes. Cada valor estará representado por el área que ocupa su porción y este tipo de gráfico puede mostrarse de tres maneras distintas, tarta, donut y piramidal. El problema principal de estas gráficas es que puede ser difícil diferenciar y comparar entre múltiples porciones de valores similares, ya que en ocasiones no es tan obvio, ni se ve a simple vista la diferencia entre ellas.

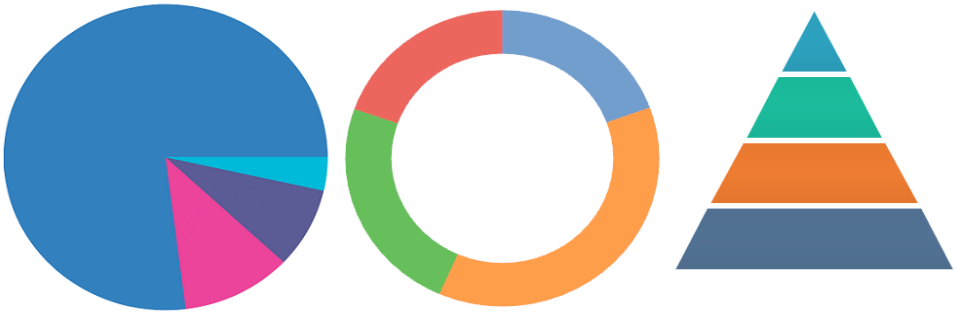

Figure 10: Gráfico de tarta, dónut y piramidal

- **Gráficas de burbujas o de dispersión** que se usan para representar tres dimensiones en un gráfico bidimensiopnal de dos ejes. La información se sitúa en unas coordenadas X e Y que representan valores mientras que la tercera dimensión se consigue a través del tamaño de las burbujas o puntos que marcan la concentración de una serie de datos.

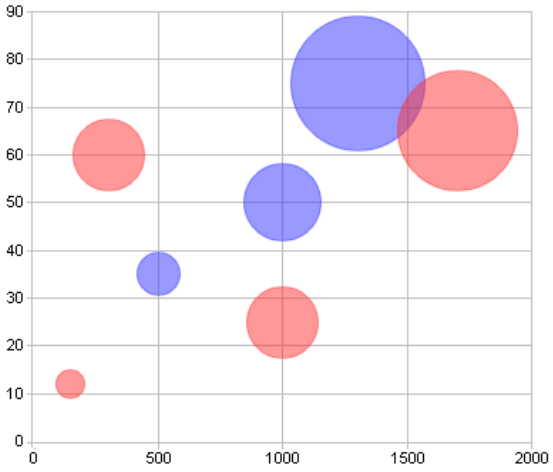

Figure 11: Gráfico de dispersión

- **Gráficos de vela/hilo/rango** que muestran el valor máximo y mínimo de un dato concreto. Son muy utilizados en el mercado bursátil, ya que al final de una jornada pueden ver rápidamente si se ha producido una gran variación en el valor de un dato concreto.

Figure 12: Gráfico de vela

- **Gráficos en forma de radar** que representan el valor de un dato en distintas categorías. Cada una de las categorías estaría representada por un eje o radio y sobre ese radio se colocaría el valor proporcionalmente según la longitud del radio. Por otro lado, cada dato deberá estar representado en cada una de las categorías hasta formar una especie de radar o telaraña.

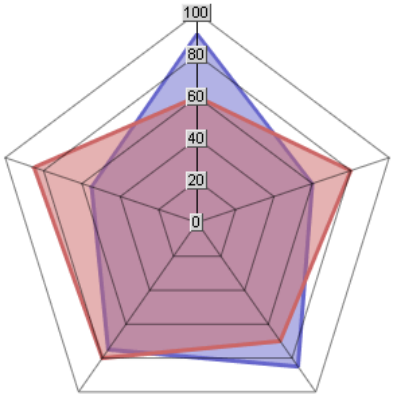

Figure 13: Gráfico de radar

2.3.2. Librerías de gráficas

Para realizar este proyecto era necesario estudiar distintas librerías de gráficas para Android, ya que son muy útiles para representar la información y los valores que se obtenga a partir de los sensores y dispositivos.

2.3.2.1. HelloCharts

La librería HelloCharts [14] para Adroid es una librería open source bajo licencia Apache v2.0 (http://www.apache.org/licenses/LICENSE-2.0) que puede encontrarse en GitHub.

Esta librería desarrolla una serie de tipos de gráficas para representar distintos tipos de datos. Las gráficas que proporciona son:

Líneas temporales → este gráfico permite añadir distintos valores de un mismo dato (puntos en un eje de coordenadas) los cuales se van agregando a una línea que muestra la evolución del mismo. Otra posibilidad que ofrece es la de poder generar múltiples líneas al gráfico, siendo cada una de ellas la evolución de un dato en concreto, de tal manera que sea muy sencillo compararlos y, en los casos que proceda, estudiar la relación que pueda existir entre ellos. Para poder utilizar este tipo de gráfica la librería proporciona la definición de un objeto punto (PointValue), el cual se va agregando a un objeto de tipo línea (Line).

Para este tipo de gráficos se permiten unas personalizaciones como la elección del color de cada línea (incluidos los puntos), hacer que los puntos sigan la distribución de una función lineal o de una cúbica, o que el área inferior a la línea se encuentre o no rellena, añadir ejes y etiquetas, etc.

Gráficos de barras → este caso es muy similar al de la línea temporal, pero cuenta con un par de funcionalidades extra. Si en el anterior caso se añadían puntos a una línea y se podían agregar múltiples líneas con sus puntos para realizar comparaciones, aquí se contarán con subcolumnas (Subcolumn) que se añadirán a las columnas (Column). La manera habitual de representar los datos será la de tener una única subcolumna por cada columna, que es precisamente el ejemplo que se ha incluido en el documento, pero al permitir añadir más de una se puede mostrar un gráfico de barras apiladas o agrupadas de tal manera que cada segmento de la columna, es decir, cada subcolumna, es proporcional a la cantidad que quiere representar dentro de ella y el valor de una columna será el total de la suma de sus subcolumnas. Por lo tanto, podrá usarse para desgranar un único valor de la columna en segmentos o partes.

Gráficas de burbujas o de dispersión → en el desarrollo para los gráficos de dispersión se tendrá algo similar a los anteriores, pero con otro tipo de objetos (BubbleValue). Una vez que se tengan generados los valores de las burbujas sólo se tendrá que instanciar un objeto de BubbleChartData que incorporará los datos que se hayan creado, a un gráfico.

Gráficos de tartas → los gráficos de tartas permiten crear secciones o porciones proporcionales de valores que se mostrarán en un gráfico con forma circular. Debido a su forma, cada porción tendrá un área gráfica proporcional a su valor dentro del área del círculo que lo contiene, teniendo en cuenta que el área total estará formada por el sumatorio de todas las porciones. En otras palabras, serán representaciones porcentuales que irán ocupando secciones del círculo de hasta completar el 100% del pastel. Además, este componente cuenta con una animación de rotación del círculo de tal manera que sea el usuario el que decida cómo quiere verlo en cada momento.

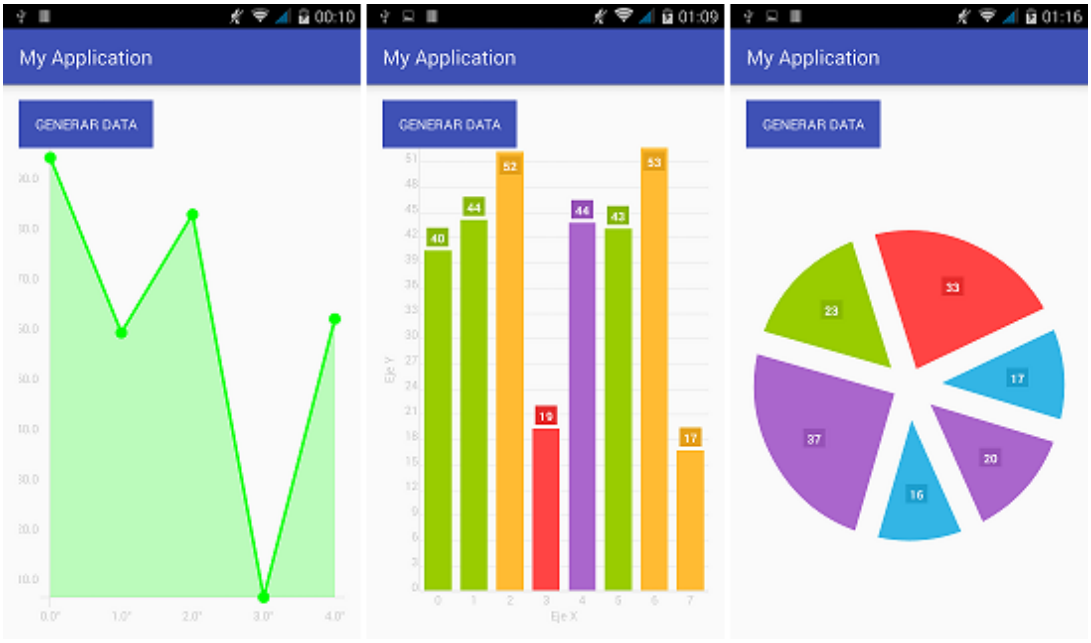

Figure 14: Ejemplos de gráficas en HelloCharts

Todos los gráficos que se visto en este punto contarán con opciones a configurar por el usuario, como por ejemplo los colores que se van a usar por cada línea, porción, subcolumna o burbuja, si se añade o no ejes al gráfico, si se utilizan etiquetas de marcado y dónde serán colocadas en caso afirmativo, etc.

Además de todo lo anterior, la librería ofrece la posibilidad de crear gráficos más complejos como por ejemplo gráficos combinados, es decir, gráficos formados por varios gráficos, vistas previas y gráficos con dependencias los unos de los otros, pero estas opciones únicamente están disponibles en la actualidad para los gráficos de barras y las líneas temporales. Otro punto de interés es que permite añadir animaciones a los gráficos para que los datos vayan apareciendo paulatinamente en la pantalla.

En resumen, HelloCharts es una librería completamente intuitiva que proporciona un listado amplio y bastante completo de gráficos de representación de datos. Contando además para cada uno de ellos con unas opciones visuales de configuración y agradables diseños lo cual lo convierte en una buena elección para el desarrollo de la aplicación.

2.3.2.2. AchartEngine

Es una librería [17] open source bajo licencia Apache v2.0. A simple vista se puede observar que los gráficos son estéticamente bastante toscos y para nada vistosos. Otro de los problemas que se han observado, es que la última actualización se hizo en 2013 por lo que parece que su desarrollo lleva abandonado bastante tiempo y no se esperan ni mejoras ni ningún tipo de arreglo, por lo que está algo obsoleta si se la compara con otras librerías de gráficas como las que se ha podido ver en puntos anteriores.

Para utilizar esta librería es necesario entender una serie de conceptos con anterioridad, lo cual implica que el usuario tiene que estudiar parte del desarrollo, ya que no se encuentra ante una librería intuitiva, sino que al inicio es algo difícil de utilizar y entender. Los conceptos de los que se hablaban en este punto son:

- La vista (View) representa el tipo gráfico que se quiere mostrar
- El conjunto de datos (Dataset) que se pintará en el gráfico (View)

- La representación (Renderer) maneja cómo se visualiza el gráfico. Bajo este concepto se controlan tanto las opciones del gráfico como las del conjunto de datos
- La factoría de gráficas se encarga de comunicar los datos con su representación para crear el gráfico

Como se puede observar, algunos de los conceptos son muy sencillos y es posible encontrar elementos equivalentes en las otras librerías. A pesar de ello y en este caso, se puede afirmar que la nomenclatura no hace muy intuitivo cómo utilizarlo y es necesario realizar numerosas pruebas hasta conseguir un resultado positivo.

Los tipos de gráficas que desarrolla la librería son los siguientes:

Línea temporal → para crear un gráfico sencillo lo que se tendrá que hacer es generar un conjunto de datos a través de sus valores en los ejes X e Y. Una vez hecho esto se deberá instanciar un objeto de representación, es decir, un Renderer para configurar las opciones gráficas y visuales de los datos como por ejemplo con qué color se van a visualizar los datos o qué ancho debe tener la línea que una los puntos. Después es necesario por un lado añadir el conjunto de datos en un nuevo objeto que es capaz de almacenar múltiples conjuntos de datos y por otro, se debe crear un nuevo Renderer que configurará el gráfico en términos de personalizar los ejes, añadir una cuadrícula, etc. Por último, se añadirá la vista al layout de Android que se esté utilizando.

Gráfico de vela o rango → para este caso se necesita crear un conjunto de datos instanciando una clase específica que genera un dataset de tipo RangeSeries que a su vez se deberá convertir posteriormente en un objeto dataset del mismo tipo del que se utiliza en las líneas temporales o los gráficos de barras.

Gráfico de tarta → en este gráfico es necesario crear cada uno de los segmentos y añadirlos a un objeto de dataset de tipo CategorySeries que como viene siendo habitual se utilizará como conjunto de datos en la vista a través de una factoría PieChartView. Además de eso, se deberá generar un Renderer para cada segmento definido en el conjunto de datos y configurarlo acorde a las necesidades del proyecto.

Gráfico de barras y gráfico de burbujas o dispersión → ambos gráficos se generarían exactamente igual que las líneas temporales con la diferencia de que al crear la vista se haría con una factoría del tipo BarChartView o del tipo ScatterChartView, pero exceptuando ese detalle son exactamente iguales que las líneas temporales. De hecho, el código es totalmente intercambiable y sólo haría falta llamar a otra función desde la factoría de creación de gráficos.

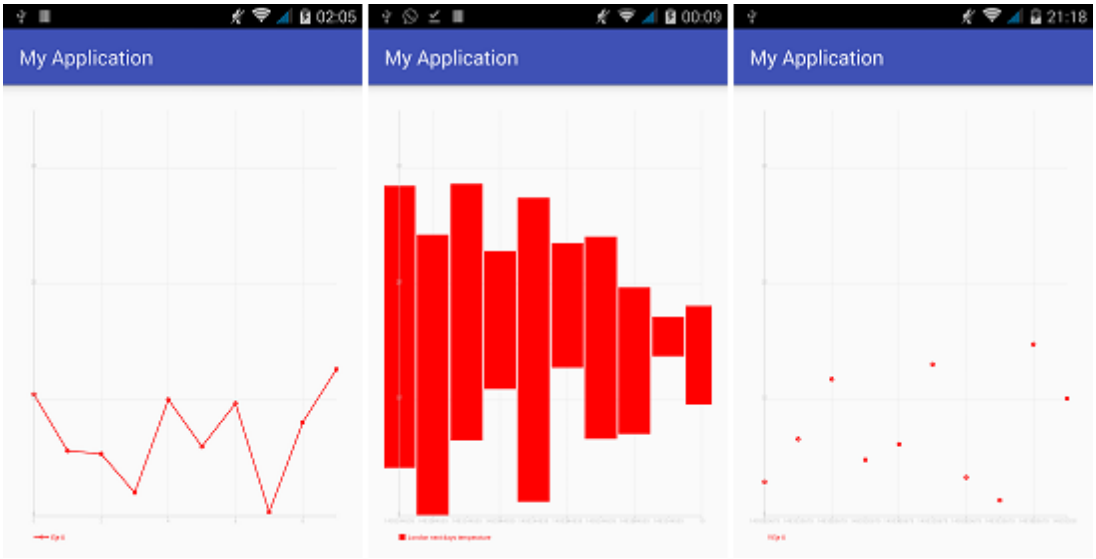

Figure 15: Ejemplos de gráficas en AChartEngine

El problema principal de esta librería es que no es nada intuitiva y cuesta bastante aprender cómo funciona, ya que cuenta con una curva de aprendizaje bastante lenta. Otra de las cosas que puede echar atrás a la hora de plantearse usarla para el desarrollo de la aplicación son pequeños detalles que se han podido ir viendo como el caso del gráfico de rango, que primero es necesario generar un tipo de datos concretos para después tener que transformarlos en el tipo de datos que se utiliza en el resto de gráficas, por lo que se añadiría un paso extra que a priori no debería existir, ya que no aporta ninguna mejora en el desarrollo, sino que provoca que el código a desarrollar sea menos legible y difícil de interpretar. Por último, hay que tener en cuenta que los resultados obtenidos con esta librería no son muy vistosos y aunque parezca algo trivial no lo es en absoluto para una aplicación Android dónde la interfaz de usuario es determinante en la mayoría de casos para fomentar o no su uso.

2.3.2.3. Williamchart

Williamchart [22] es otra librería de gráficos para aplicaciones Android. Es una librería muy sencilla que se centra en la visualización de la información de forma intuitiva y con una estética agradable. No proporciona múltiples opciones de configuración como pueden ofrecer otras librerías, pero es algo hecho a propósito porque el autor considera que es innecesario y que es más importante mantener la librería lo más simple y limpia posible. A pesar de ello, sí que ofrece algunas opciones interesantes y útiles para los desarrolladores. Estas opciones las divide en dos grupos, las opciones que afectan a todos los gráficos en general como por ejemplo el hecho de tener o no ejes y las más concretas para cada uno de los gráficos como redondear las esquinas de las barras del gráfico de barras.

Además de esto, la librería es open source y está bajo una licencia Apache v2.0 al igual que otras que se han visto con anterioridad por lo que permite no sólo el uso de la librería, sino también modificaciones para que ajuste a aquello que se necesite.

Los tipos de gráficas que desarrolla la librería son los siguientes:

Línea temporal → en esta librería existe un tipo de datos o dataset específico para cada tipo de gráfico. Es decir, que los datos que se utilicen en un tipo de gráfico no pueden ser reutilizados en otro porque no son equivalentes. Para las líneas temporales se crearán múltiples puntos que contarán con un valor y un etiquetado, estos puntos se añadirán a una línea que posteriormente será agregada a un gráfico del tipo LineChartView que a su vez extiende de la clase ChartView como el resto de gráficas de la librería.

Gráfico de barras/Gráfico de barras horizontal → en los gráficos de barras se contaría con un nivel

menos de complejidad, ya que sólo tendría barras y serán ellas mismas las que representen el valor del dato. En el caso del gráfico en posición horizontal, se podrá generar el mismo conjuntos de datos (que en ambos casos será un BarSet), pero en lugar de instanciar un objeto BarChartView, se instanciará un HorizontalBarChartSet. Las opciones que puedan configurarse en uno de ellos serán igual de válidas para el otro.

Gráfico de barras en modo apilado-agrupado/Gráfico de barras en modo apilado horizontal → para poder agregar un gráfico de barras agrupado se actuaría exactamente igual que para los gráficos de barras simples, eso sí, intercambiando el valor que se agregaba por un array de valores. Esto será así porque cada BarSet necesitara cada una de las agrupaciones para poder representar la información, ya que cada una de las secciones serán partes de un todo que será precisamente una barra.

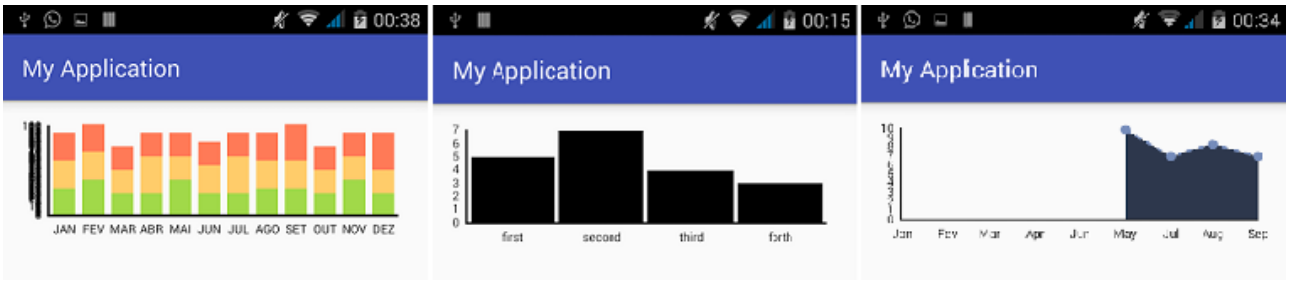

Figure 16: Ejemplos de gráficas en WilliamChart

Un detalle bastante original que tiene esta librería es la posibilidad de animar los datos en el eje Y, de tal manera que aparezcan de abajo a arriba, hasta alcanzar su valor real dentro de esa dimensión. Las animaciones pueden definirse tanto con una franja de tiempo en la que se reproduzca el efecto como con una función que lo suavice.

Como es posible observar, esta librería ofrece muchas menos posibilidades de representar la información que el resto, ya que en realidad es como si sólo tuviera tres gráficos distintos. Las versiones horizontales aunque visualmente agradables e interesantes no aportan nada que no lo hagan sus versiones en vertical, mientras que el gráfico de barras y el gráfico de barras apilado tampoco es que proporcionen mucha más información ni una representación necesaria o esencial para este caso teniendo en cuenta el tipo de datos que se van a manejar en la aplicación.

2.3.2.4. MPAndroidChart

MPAndroid [15] es una librería de Android para crear gráficas. La librería se encuentra bajo una licencia Apache v2.0 como el resto que se han apuntado en este apartado, por lo que además de ser open source permite la modificación de la misma para así ajustarse a las necesidades del proyecto. El punto fuerte de esta librería, ya que no difiere mucho de las anteriores que se han observado (sobre todo de HelloCharts), son sus fluidas animaciones, que aunque no son especialmente útiles y en algunos casos pueden distraer al usuario le da un aspecto muy dinámico a los gráficos. El único problema que podría haber con las animaciones es que sólo son compatibles con versiones de la API de Android 11 o superior, pero leyendo el gráfico de market share de Google, es posible observar que cubriría más de un 90% de los dispositivos actuales, así que realmente no es un punto negativo a tener en cuenta.

Esta librería permite crear los siguientes tipos de gráficas:

Líneas temporales → para crear este tipo de gráfico, se debe generar un listado de registros que compondrán el conjunto de datos. Después, ese listado de registros tendrá que añadirse a un objeto que representará la línea temporal (LineDataSet) de tal manera que sea posible personalizar toda una línea conjuntamente. En este paso serían capaces de cambiar colores, añadir las etiquetas de leyenda, etc. Por último, se relacionará cada una de las líneas temporales que hayan sido definidas con el gráfico (LineData) al que pertenecen.

Gráficos de barras (y su versión en horizontal) → para este caso se contará con el mismo proceso que para las líneas temporales, pero en lugar de utilizar un listado de "Entry" se utilizará "BarEntry", "BarDataSet", etc. Para este gráfico se deberá contar también con tantas etiquetas de leyenda como valores se tengan, ya que los usará para ir marcando y ofreciendo información al usuario de qué representa cada una de las barras.

Gráficos de burbujas y dispersión → se basa en los mismos procesos que se han visto anteriormente.

Gráfico de tartas → para los gráficos de tartas se debe añadir cada una de las entradas con dos parámetros. El primer parámetro representa el valor del dato y el segundo parámetro debe ser un índice incremental que represente el total de secciones o segmentos en los que se dividirá la tarta. Por supuesto, cada uno de los segmentos deberá tener su propia etiqueta que defina su contenido, ya que si no, no se podría interpretar el gráfico correctamente. Por último y como viene siendo habitual, los colores de cada uno de los segmentos (al igual que de cada una de las barras) pueden ser personalizables.

Gráficos de velas → este tipo de gráfico es un poco más complejo, ya que son necesarios cinco valores distintos para cada uno de los datos. El primero de los valores representa el índice del dato en el eje Y, los siguientes representan el valor más alto, el más bajo, el valor con el que comienza el rango del valor bursátil y el valor con el que se cierra el rango.

Gráficos de tipo radar → esta es la única librería de las que se han analizado hasta el momento que permite este tipo de gráfico. Cada una de las áreas del radar representará un conjunto de entradas concreta, para cada conjunto, se deberá tener el mismo número de entradas, ya que definirán los radios del radar para todos los conjuntos de datos.

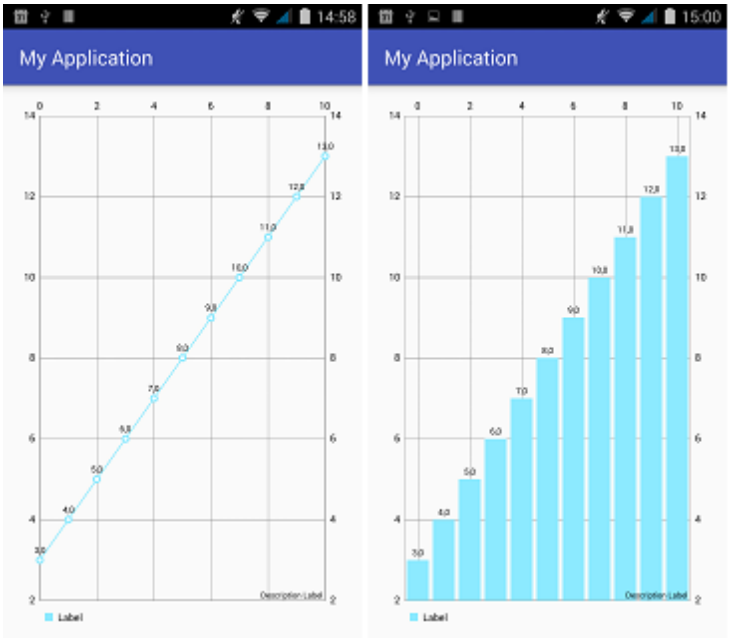

Figure 17: Ejemplos de gráficas en MPAndroidChart

Para cada modificación de los datos de cualquiera de los gráficos deberá ser necesario la llamada al método invalidate() para que refresque la visualización del mismo. Es necesario especificar que no sólo es necesario para actualizar los valores de los datos que se muestren en la gráfica, sino que también será absolutamente necesario para cualquier otro cambio de estilos, ya sea un color de fondo, mostrar la cuadrícula o no, etc. Si este detalle no se tiene en cuenta, no se mostrarán los efectos.

Las animaciones que proporciona pueden afectar únicamente a uno de los ejes o a ambos a la vez para que los valores vayan apareciendo dinámicamente en el layout del usuario. Si se animara

únicamente el eje X, la animación representará la aparición de los datos de forma escalonada siguiendo un tiempo concreto, de tal forma que hasta que no pase, no podrán visualizarse todos los datos. En caso de ser el eje Y el que se decida a animar, sí que se contarán con todos los datos desde el inicio, pero alcanzarán su valor real al finalizar el tiempo definido para la animación. Las curvas de velocidad, es decir, las funciones de suavizado son las habituales de otros módulos de animaciones como la lineal, la cúbica, etc.

En resumen, esta librería permite un gran número de opciones y posibilidades de personalización. Su punto de diferenciación frente a otras librerías es el desarrollo de las animaciones que están muy conseguidas y la posibilidad de crear un número mayor de gráficos. En su contra tiene que en algunos casos es menos intuitiva que otras librerías y que es algo más rígida a la hora de crear los gráficos.

2.3.2.5. Android Plot

Android Plot [21] es una librería open source bajo licencia Apache v2.0 cuyo código puede encontrarse y estudiarse en la plataforma Bitbucket. Al igual que en el resto de librerías, el gráfico se añade desde el propio XML y posteriormente en la actividad se debe añadir el conjunto de datos que se van a cargar en él.

Los tipos de gráficos que permite son los siguientes:

Líneas temporales → para el gráfico y como ocurre en la mayoría de librerías genera un conjunto de datos compuestos por los valores X e Y que representarán los puntos de las líneas en sus coordenadas. En este caso, el valor X de la serie no hará falta concretarlo y lo deducirá de la posición del punto en el array, así que ese eje será incremental de unidad en unidad. Después, el usuario deberá definir la visualización del mismo, definiendo los colores para los puntos, la línea y el área que comprende. Además, también definen la suavidad de la curva de unión de la línea temporal. Para esta parte de personalización de estilos se instancia un nuevo objeto Formatter en el que es posible acceder a todas las opciones de configuración del gráfico. Por último, hay que relacionar el objeto Formatter con el conjunto de datos.

Gráficos de barras → para poder crear esta gráfica será necesario seguir los pasos de la línea temporal y será a través del objeto de la clase Formatter cómo se definirá que se trata en realidad de un gráfico de barras. Por un lado este desarrollo es totalmente reutilizable, porque es posible usar una y otra vez los mismos conjuntos de datos, únicamente relacionándolo con un Formatter específico u otro, así que el parseo de datos es completamente trivial. El problema es que la estructura de los Formatter es poco intuitiva para su uso y que es necesario instanciar un nuevo objeto para poder generar el gráfico.

Gráficos de tartas → al igual que en el resto de casos, se necesita un Formatter propio que defina los colores de la barra y de los bordes, además de usarse para poder concretar si se pinta en la pantalla un gráfico de tarta o de tipo donut. Para la generación de los datos se definen segmentos y es la propia librería la que calcula qué área le corresponde a cada uno de ellos hasta formar con el total el porcentaje correspondiente.

Gráficas de burbujas o de dispersión → los gráficos de tipo burbuja en esta librería crean una serie o conjunto de datos compuesto de tres elementos para cada dato que serán los valores de coordenadas X e Y con un valor Z que representa el radio de la burbuja.

Gráficos de vela/hilo → se deben definir cuatro arrays que representen los valores máximos, mínimos, de apertura y clausura de cada dato bursátil, es imposible crear este gráfico sin contar con los cuatro tipos de valores. Al tener los cuatro arrays el usuario se tiene que asegurar de que todas tengan el mismo tamaño y que se incluyan precisamente en el orden que se ha expuesto en este mismo párrafo.

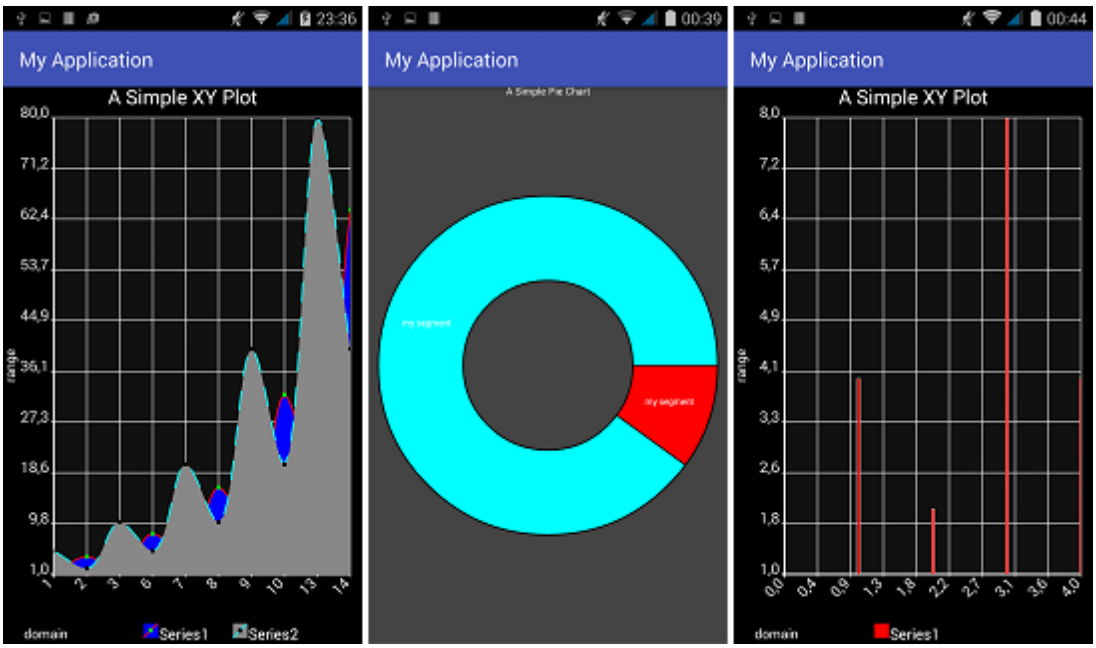

Figure 18: Ejemplos de gráficas en AndroidPlot

Esta librería es bastante difícil de interpretar en un inicio, y a pesar de haber estado probando con ella distintos tipos de gráficos no es sencillo sentirse cómodo con ella porque es muy poco homogénea y son necesarias muchas líneas de código para llegar a crear una única gráfica, haciendo que se vea confuso y poco limpio. Por otro lado, las opciones de configuración de estilos utilizan nomenclaturas que son difíciles de entender y de seguir por lo que había que estar continuamente revisando la documentación para saber qué uso tenía cada opción y cómo es posible aplicarlo, ya que si no se cuenta con el Formatter no se podrá pintar el gráfico.

Además y cómo se puede observar en la imagen adjunta, los diseños no son nada agradables a la vista, parecen antiguos y poco trabajados que es un detalle muy a tener en cuenta para el proyecto.

2.3.2.6. Syncfusion Chart

La empresa detrás del desarrollo de Syncfusion Chart [19] permite la adquisición de una Community License gratuita a imagen y semejanza de la que proporciona Microsoft con VisualStudio Community 2015. Eso sí, para poder hacerse con ella se deben cumplir una serie de condiciones. La librería es una solución no open source, lo cual es un punto en su contra, ya que de esta manera es más difícil entender algunas funcionalidades complejas que pueda contener, aun así, se ha decidido probar su uso para comprobar si es posible incorporarla al proyecto de manera satisfactoria y cumplir todas y cada una de las necesidades que existen en el desarrollo.

La librería permite crear los siguientes tipos de gráficas; Líneas temporales, gráficos de barras, gráficos de tartas/donuts/piramidales, gráficas de burbujas o de dispersión, gráficos de vela/hilo y gráficos en forma de radar. Es decir, que proporciona un paquete de tipos de gráficas bastante amplio, aunque tampoco aporta nada nuevo que no se haya visto con anterioridad.

En realidad, la librería utiliza por debajo la librería de Xamarin.Forms, que es una API para crear aplicaciones nativas en Ios, Android y Windows Phone con el lenguaje C#. De hecho, las propias librerías para Android son DLLs compiladas, que por supuesto no es posible estudiar, ya que no permiten a acceder al código sin compilar en ningún otro lado.

Debido a esto y a la imposibilidad de recompilarla a otro formato que sea compatible con Java y que pueda incorporarse en Android Studio, sólo se ha sido capaz de revisar a través de la documentación proporcionada y de tutoriales cómo funciona realmente. Mientras que Syncfusion Chart se encarga

de la creación del código de los distintos tipos de gráficas en C#, Xamarin.Forms transformará dicho código en código nativo de la aplicación Android. Así, ambos paquetes trabajarán mano a mano para crear la aplicación, lo cual no podría ser posible si cualquiera de las dos faltara.

2.3.2.7. Otras librerías

Además de estas librerías que se han estado probando y exponiendo en esta sección, merece la pena mencionar la existencia de muchas otras que también han sido ojeadas, aunque no se haya profundizado tanto en su estudio, ya que no se ha visto que tengan tantas posibilidades para poder ser utilizadas en este proyecto. Dentro de este grupo se encontrarán tanto Holo Graph Library [16] que aunque muy limitada en sus gráficos y sus opciones tiene unos diseños muy actuales, como SciChart [18] que es un software propietario bajo licencia propia y cuyo código original es inaccesible. Esta última cuenta con infinidad de tipos de gráficos, aunque también es cierto que muchos de los que nombra en su documentación son versiones de otros gráficos con algún dato extra o similar. También es un hecho muy interesante que se trata de una librería que sigue en continua evolución y va incorporando paulatinamente nuevas gráficas y otros elementos relacionados. Por último, también se ha estudiado Charts4j [20], ésta es una librería Java que puede utilizarse desde Android para la representación de datos. Cabe reseñar que además de estar desarrollada íntegramente en Java, admite el uso del paquete de herramientas Google Chart Tools, de tal manera que los gráficos que se generen a partir de ella, en realidad, se generarán a través de la API del paquete de Google.

2.3.2.8. Comparativa

Como resultado de este estudio de librerías, se ha llegado a la conclusión de que la librería que más se ajusta a las necesidades del proyecto es HelloCharts, ya que por un lado y aunque no es la librería que contiene el mayor número de gráficos distintos (esa sería MPAndroidChart), sí que contiene aquellos que van a ser útiles y que se necesitán en la aplicación a desarrollar, mientras que por otro lado es la librería más intuitiva entre el total de las mismas y su facilidad de uso es un punto clave en la elección. Esto se debe a que al finalizar el trabajo se deberá estar completamente familiarizado con la librería, ya que va a necesitarse en todas las fases del desarrollo. Esa familiarización que se ha conseguido durante el estudio es lo que ha hecho que la balanza se decante a su favor.

Además de esos detalles, los diseños utilizados y su desarrollo posterior son estéticamente bastante atractivos para los usuarios, ya que cuentan con un estilo moderno y original, con colores suaves, separaciones abundantes y suficientes entre los elementos, etc. Como ya se ha comentado, este detalle no es para nada trivial en este desarrollo, así que también ha sido determinante.

Por todos estos puntos, la mejor elección para este proyecto es la librería HelloCharts.

2.4. Android

Además de investigar y estudiar las librerías disponibles es también una tarea importante estar al tanto de las características con las que cuenta Android para poder tomar la decisión más adecuada a la hora de diseñar la solución técnica.

Lo primero que se debía tener en cuenta, era la estructura de un proyecto básico en Android. Es importante que aunque un proyecto de Android pueda contar con múltiples aplicaciones, también llamadas módulos, en la práctica se suele desarrollar una única aplicación por proyecto. Esto quiere decir, que habrá configuraciones que se puedan gestionar tanto a nivel de proyecto como de aplicación o de ambos y en el caso en el que se defina en ambos ámbitos, la configuración del módulo tendrá prioridad frente a la del proyecto.

En las aplicaciones Android es necesario conocer tanto las opciones gráficas como las opciones de programación con las que se cuentan a la hora de desarrollar un proyecto.

2.4.1. Características gráficas

Dentro de las opciones gráficas, el módulo de "Layout" contiene todas las vistas que se utilizan en la aplicación. Desde las vistas completas de páginas o la del menú principal hasta cada uno de los fragmentos de vista que se inserten en cada una de ellas.

Los layout son interfaces de usuario que contienen elementos de control, también llamados elementos de diseño debido a su propia naturaleza gráfica. En Android es posible definir distintos tipos de layout según las necesidades de disposición de los elementos de control, pero todos y cada uno de ellos pueden contener todos los tipos de elementos de diseño. En los siguientes puntos se enumerarán, para que cuando se hable de ellos posteriormente, esté claro cómo deberían situarse sus contenidos.

Para todos los tipos de layout se incluirá un ejemplo de cómo se visualizarían una serie de elementos según el código XML y el layout escogido.

La imagen que se muestra a la izquierda es un ejemplo de los controles que se añadirán en los layout.

Figure 19: Ejemplo controles

- FrameLayout → es el layout más básico de todos. Coloca sus elementos en la esquina superior izquierda y unos superpuestos a otros, por lo que si todos tuvieran el mismo tamaño únicamente se visualizaría el último. Debido a esto, este layout suele utilizarse únicamente con un único elemento en su interior. En la imagen de abajo es posible ver cómo se ocultarían parte de los controles debido al orden de la definición.

FrameLayout

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.and
    xmlns:tools="http://schemas.android.com/to
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="50px"
    tools:context="com.test.miriam.myapplicati
    <Button
        android:layout_width="300px"
        android:layout_height="100px"
        android:id="@+id/id_a"
        android:background="@color/Yellow" />
    <Button
        android:layout_width="400px"
        android:layout_height="50px"
        android:id="@+id/id_b"
        android:background="@color/Red" />
    <Button
        android:layout_width="500px"
        android:layout_height="250px"
        android:id="@+id/id_c"
        android:background="@color/Green" />
    <Button
        android:layout_width="150px"
        android:layout_height="400px"
        android:id="@+id/id_d"
        android:background="@color/Blue" />
</FrameLayout>
```

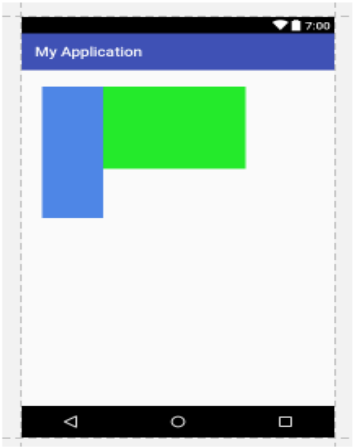

Figure 20: Posición de controles en un FrameLayout

- LinearLayout → en este tipo de layout los elementos se colocan de manera lineal uno a continuación del otro. Eso sí, el layout permite decidir si el orden de aparición será basándose en una orientación vertical u horizontal. Para que esto quede un poco más claro, si se escoge orientación vertical, los elementos se colocarán de arriba a abajo mientras que si se escoge horizontal, se presentarán de izquierda a derecha. En el caso de la imagen se ha escogido orientación vertical y al no haber añadido ningún tipo de margen entre ellos, los controles aparecen completamente pegados, pero no solapados. En este caso no se ha querido añadir un ejemplo con elementos con pesos (dimensiones) relativas, ya que se ha preferido dejarlas fijas, pero más abajo se explicará el uso de esta propiedad.

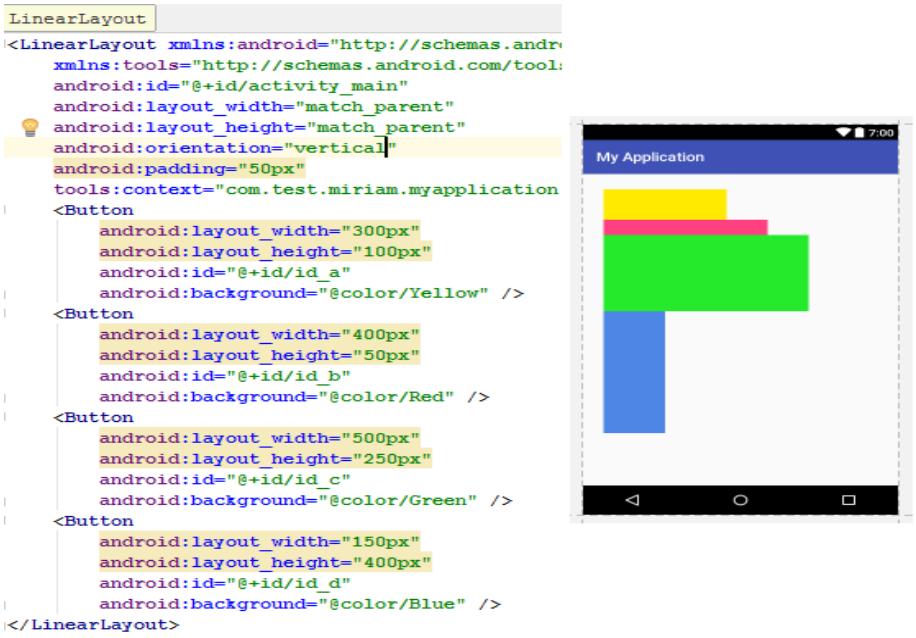

Figure 21: Posición de controles en un LinearLayout

- RelativeLayout → es un layout que tiene exactamente las mismas características que el LinearLayout, pero que además justo como indica su nombre, permite situar los elementos relativos a otros elementos. Dentro de las opciones de posicionamiento se cuenta con propiedades como android:layout_below que posiciona la zona superior del control que tiene esta propiedad justo debajo del elemento cuyo ID se haya indicado, como android:layout_toRightOf que posiciona el borde izquierdo del control que tiene esta propiedad justo pegando al borde derecho cuyo ID se haya indicado o como android:layout_centerVertical que centra el elemento verticalmente según la altura del padre, y estas son solo algunas de las posibilidades. Para verlas todas, basta con acceder a la documentación de Android, pero como se ha podido observar con este par de ejemplos siempre son posiciones relativas a otros elementos, en algunos casos elementos padre y en otros hermanos. En la imagen que se ha creado es posible observar como unos elementos se colocan y ordenan de acuerdo a otros. El único de ellos que no tiene definido ningún tipo de posición relativa es el que tiene identificador id_d o lo que es lo mismo el componente azul, es por esta razón y no otra que se coloque al inicio del layout y que el resto de elementos se sitúen relativamente a este. Después se puede observar que tanto el elemento amarillo como el elemento rojo deben colocarse debajo del elemento azul, ya que utilizan la propiedad layout_below con el identificador id_d debido a esto ambos elementos se superponen, pero como el control rojo tiene también la propiedad layout_alignParentRight se alinea el elemento a la derecha de su elemento padre, que es el propio layout relativo. Si el elemento no está pegado al borde, se debe a que se ha añadido en todos los ejemplos de los tipos de layout un

padding de 50 píxeles que los separe de las esquinas, ya que se ve algo peor sin esos valores.

```
RelativeLayout  Button
<RelativeLayout xmlns:android="http://schemas.a
    xmlns:tools="http://schemas.android.com/too
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="50px"
    tools:context="com.test.miriam.myapplicatio
    <Button
        android:layout_width="300px"
        android:layout_height="100px"
        android:id="@+id/id_a"
        android:layout_below="@id/id_d"
        android:background="@color/Yellow" />
    <Button
        android:layout_width="400px"
        android:layout_height="50px"
        android:id="@+id/id_b"
        android:layout_below="@id/id_d"
        android:layout_alignParentRight="true"
        android:background="@color/Red" />
    <Button
        android:layout_width="500px"
        android:layout_height="250px"
        android:id="@+id/id_c"
        android:layout_below="@id/id_a"
        android:background="@color/Green" />
    <Button
        android:layout_width="150px"
        android:layout_height="400px"
        android:id="@+id/id_d"
        android:background="@color/Blue" />
</RelativeLayout>
```

Figure 22: Posición de controles en un RelativeLayout

- TableLayout → para los TableLayout es necesario definir previamente el número de filas y columnas que lo compondrán. Para ello se definirán etiquetas que representarán cada una de las filas, llamadas TableRow, y las columnas serán igual al mayor número de elementos que se encuentren definidos dentro de las filas. A partir de este punto se podrán ir situando cada uno de los elementos en cada una de las celdas de la tabla, pero no es necesario que todas las celdas tengan algún valor ni que todas ocupen el mismo espacio. Por defecto, el tamaño que se dará a cada columna será igual al mayor ancho entre los elementos que conformen la misma, pero existen propiedades que pueden modificar este comportamiento para ocultar alguna columna, agrandarla o disminuirla o en lugar de hacerlo sobre una columna completa, se puede aplicar a una celda de manera individual para que ocupe más de una columna. En el ejemplo que se ha creado se han añadido dos TableRow o lo que es lo mismo, dos filas y dentro de cada una de ellas se han incluid dos controles creando así dos columnas en cada una de las filas. Debido a las dimensiones de los propios controles no son capaces de colocarse uno a continuación del otro sin que se salgan del layout de ejemplo y este es un problema que ocurre en ambas filas, así que el contenido sobrante simplemente no se pinta.

```
TableLayout
<TableLayout xmlns:android="http://schemas.android.
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="50px"
    tools:context="com.test.miriam.myapplication.Ma
    <TableRow>
        <Button
            android:layout_width="300px"
            android:layout_height="100px"
            android:id="@+id/id_a"
            android:background="@color/Yellow" />
        <Button
            android:layout_width="400px"
            android:layout_height="50px"
            android:id="@+id/id_b"
            android:background="@color/Red" />
    </TableRow>
    <TableRow>
        <Button
            android:layout_width="500px"
            android:layout_height="250px"
            android:id="@+id/id_c"
            android:background="@color/Green" />
        <Button
            android:layout_width="150px"
            android:layout_height="400px"
            android:id="@+id/id_d"
            android:background="@color/Blue" />
    </TableRow>
</TableLayout>
```

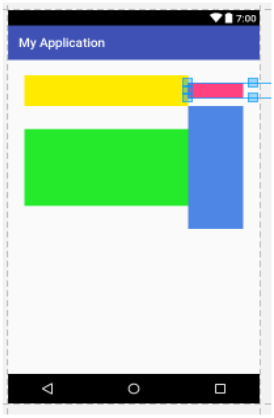

Figure 23: Posición de controles en un TableLayout

- GridLayout → para el caso de GridLayout se tendrá algo similar a lo que era posible ver con el TableLayout y hasta se podría decir que es una evolución del mismo, pero tiene algunas variaciones en cuanto a la disposición de los elementos. En este caso el número de filas y columnas no vienen definidas por los elementos que contiene el layout, sino que se definen a través de unas propiedades específicas. De esta manera, el desarrollador se ahorra las etiquetas TableRow que se veían en el punto anterior, aunque sigue siendo capaz de incluir las propiedades que modificaban el ancho de una celda concreta como si de una tabla se tratase. En este ejemplo de GridLayout se ha definido que debe mostrar una cuadrícula que cuente con dos filas y tres columnas. Por ello y al tener únicamente cuatro controles, los tres primeros se colocarán en la primera fila y el cuarto y último elemento se mostrará en la segunda. Debido al tamaño fijo de los propios elementos el tercer componente casi no puede visualizarse y se sitúa por fuera del terminal al igual que ocurría en el TableLayout.

```
GridLayout  Button
<GridLayout xmlns:android="http://schemas.android.co
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="50px"
    android:rowCount="2"
    android:columnCount="3"
    tools:context="com.test.miriam.myapplication.Mai
        <Button
            android:layout_width="300px"
            android:layout_height="100px"
            android:id="@+id/id_a"
            android:background="@color/Yellow" />
        <Button
            android:layout_width="350px"
            android:layout_height="50px"
            android:id="@+id/id_b"
            android:background="@color/Red" />
        <Button
            android:layout_width="500px"
            android:layout_height="250px"
            android:id="@+id/id_c"
            android:background="@color/Green" />
        <Button
            android:layout_width="150px"
            android:layout_height="400px"
            android:id="@+id/id_d"
            android:background="@color/Blue" />
</GridLayout>
```

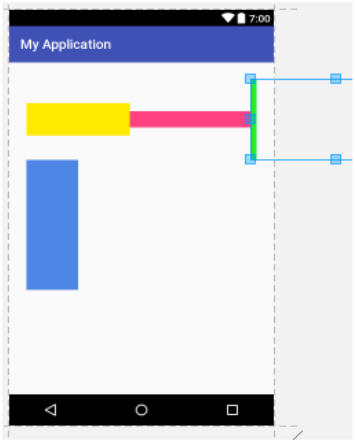

Figure 24: Posición de controles en un GridLayout

- DrawerLayout → son layouts utilizados para pintar vistas interactivas que se deslizan desde los lados de la ventana de la aplicación, así que como es de suponer sólo se podrá contar con un máximo de dos DrawerLayout, uno para que salga desde el borde izquierdo y otro para que salga desde el derecho. Debido a estas limitaciones y especificaciones, será un layout que se utilizará mayormente para generar menús.

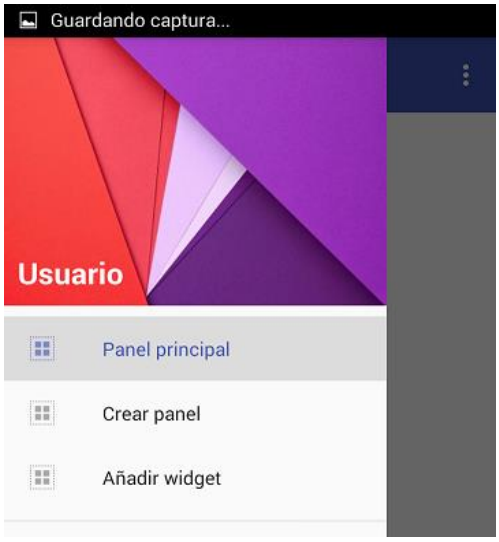

Figure 25: Ejemplo de un DrawerLayout

Algunas de las propiedades que se pueden añadir a cualquiera de los layouts son la de definir un ID con "android:id", que es una propiedad válida para todos los elementos de los layout, determinar el ancho y el alto que debe ocupar en el dispositivo con "android:layout_width" y "android:layout_height" que puede tomar los valores match_parent o match_content dependiendo de si se quiere que ocupe el 100% del espacio disponible en la pantalla o si por el contrario se prefiere

que se ajuste a su contenido. También es posible introducir una medida de ancho y alto fija, añadiendo la cantidad y la medida en la que debe procesarse, como por ejemplo 100px. Para el dimensionamiento de los elementos existe otra propiedad que puede usarse en los layouts llamada "android:layout_weight". Esta propiedad dimensiona los elementos de control con medidas proporcionales de tal manera que cada elemento tendrá un peso concreto en la pantalla y dependerá de lo que ocupen los demás, pero siempre teniendo en cuenta que debe repartirse entre todos los elementos el tamaño total de la pantalla. En otras palabras si se tiene un elemento de control TextView con peso igual a 1 y un elemento Button con peso 2, el segundo elemento ocupará el doble de lo que ocupe el primero, siempre y cuando entre ambos elementos ocupen el 100% del espacio como si se les aplicará conjuntamente el valor match_parent. Por supuesto, otras propiedades visuales como márgenes, paddings, etc. También pueden aplicarse a estos componentes como si se tratara de un elemento de diseño más.

Además, para todos los tipos de layout, es posible agregar un elemento ScrollView que los englobe. El ScrollView es un elemento propio de Android que permite la visualización de contenido de mayor tamaño que la pantalla del dispositivo. Para ello es importante fijar las propiedades del tamaño de ancho y de alto porque se deberá limitar si únicamente se cuenta con scroll horizontal (en cuyo caso se debería utilizar el elemento HorizontalScrollView) con scroll vertical o con ambos.

Ahora se van a definir algunos de los elementos de control o controles que se han mencionado en los párrafos anteriores. Los controles son elementos de diseño que se sitúan dentro de los layouts para renderizarse en las vistas de la aplicación. Estos elementos pueden crearse directamente en el layout a través de etiquetas o incluirse de manera programática desde el código Java. A continuación, se enumerarán los controles más conocidos para que sea más fácil entender de qué se está hablando.

En Android existen infinidad de controles que añadir a los distintos layouts por lo que el documento no se centrará en explicar cada uno de ellos porque no se acabaría nunca. No obstante, sí que es importante enumerar algunos de los más usados para que sea posible hacerse una idea de qué se está hablando:

- Botones
  - Button y ImageButton son dos controles de tipo botón que se diferencian en que mientras que uno contiene un texto, el otro contiene una imagen
  - ToggleButton y Switch son dos controles cuya diferencia es gráfica. Ambos son botones que permiten incluir un texto distinto dependiendo de si están pulsados o no
  - FloatingButton es un tipo de botón que surgió a partir de la filosofía de diseño y usabilidad de Material Design y que en la actualidad se ha incorporado en muchísimas aplicaciones a través del uso de la librería de soporte de diseño versión 22.2.0 que se tendrá que incluir en el Gradle si quiere usarse con la llamada 'com.android.support:design:22.2.0'
- Textuales
  - TextView es un elemento textual que contiene una cadena de caractéres
- Formularios y recogida de datos
  - EditText define el clásico campo de entrada textual
  - Checkbox y Radiobutton son controles en los que se seleccionan opciones, aunque mientras que en el checkbox pueden seleccionarse de 0 a X valores, en los radiobutton sólo es posible escoger una única opción de entre todas. Con estos elementos es posible desde el código comprobar su estado y realizar acciones según los resultados de las mismas, es decir, que estarán atentos a los eventos que afecten a las vistas
- Listados

- Spinner o la lista desplegable es un control más complejo que contiene un listado de valores
- ListView que es un listado vertical de elementos, siendo los elementos un layout propio que puede a su vez contener otros controles. Podría decirse que es un listado de objetos con atributos propios
- GridView crea una vista en forma de tabla con un listado de elementos ordenados de izquierda a derecha y de arriba a abajo

- Otros
  - ImageView se utiliza para la carga de imágenes

Aquí se muestra una imagen de ejemplo de distintos tipos de controles que se ha obtenido de la documentación oficial de Android.

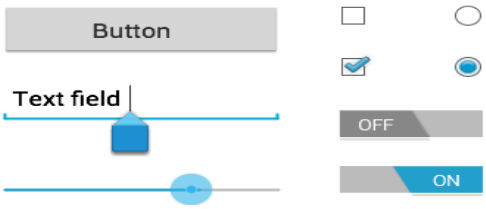

Figure 26: Ejemplo de controles Android

Como se ha comentado, existen muchos otros controles, pero no merece la pena enumerarlos todos porque no es solo que no hayan sido usados en este proyecto, sino que en general son elementos más complejos que se usan en casos concretos.

2.4.1.1. Otras características de diseño

Dentro de las características gráficas, aparte de la utilización de layouts, la aplicación puede personalizarse a través del módulo "Values" y de sus ficheros. Este elemento contendrá todo tipo de constantes, estilos y literales que se definan para la aplicación.

En otras palabras, permite definir todos los ficheros de recursos que se necesiten por lo que será útil categorizar los tipos:

- colors.xml → guardará los valores de las variables de estilos de color
- dimens.xml → almacenará los tamaños de los elementos con sus correspondientes unidades de medida. Las unidades de medida podrán ser diversas, por ejemplo píxeles, centímetro, etc. Y no es necesario que todas utilicen el mismo estándar
- string.xml → es el fichero que contiene todas las etiquetas textuales de literales de la aplicación. Este fichero será el que se intercambiará
- styles.xml → determina el estilo de la interfaz de usuario o vista. Es decir, especifica la apariencia gráfica y puede aplicarse tanto a nivel de layout como a nivel de control

Uno de los puntos fuertes de utilizar estos ficheros "Values" durante el desarrollo de los "Layout" es que al tener los valores centralizados en un mismo punto, si el proyecto requiere un cambio de tamaños, colores, textos, etc. Sólo será necesario cambiarlo en uno de estos ficheros en lugar de tener que cambiar el valor en cada una de las vistas, con la cantidad de tiempo que conllevaría hacerlo. Si estos ficheros no existieran o no se usaran, crear una versión de una aplicación en otro idioma o hacer algún cambio simple de estilos significaría tener que realizar una revisión global de la aplicación para

tener que cambiar todos elementos que tuvieran relación o se tuvieran que ver afectados por el cambio, por ejemplo cambiar el tamaño de todos los títulos de secciones supondría una única modificación. Es decir, que la mayor ventaja es la centralización de recursos y su facilidad de acceso.

Para el uso de estos ficheros desde los "Layout", solo es necesario apuntar a la ruta del valor que se quiere utilizar, por lo que se necesitará decir en qué fichero de "Values" se encuentra el recurso y cuál es el identificador del mismo. Un ejemplo de uso de los ficheros "Values" es:

```
android:text="@string/addWidget"
```

Mientras que para el uso de estos ficheros programáticamente, es decir, desde las clases "Java", se utilizará la función getString con el identificador de la cadena de caracteres exactamente como se muestra en la siguiente línea:

```
String string = getString(R.string.addWidget);
```

Pero lo más interesante del desarrollo de ficheros "Values" es que se pueden definir distintos ficheros según características del propio terminal móvil que utilice la información. Para ello, únicamente se tiene que crear una carpeta cuyo nombre comience con "values-" y a continuación el parámetro del terminal con el que se quiere comparar. De esta manera se tendrán unos valores por defecto para todos los dispositivos y luego se crearán los archivos con las excepciones como por ejemplo utilizar un color específico para un terminal que tenga una SDK 14 utilizando el directorio "values-v14" o definir un tamaño distinto de textos para dispositivos con una pantalla cuya resolución sea al menos de 720 píxeles y que además estén en posición landscape, con la carpeta "values-sw720dp-land", landscape significará en este contexto que su orientación es horizontal.

2.4.2. Características de programación

En Android, podría decirse que la unidad básica de un proyecto es un Activity. Según la documentación oficial un Activity es un componente que contiene una pantalla con la que los usuarios pueden interactuar para realizar una acción.

Las clases Activity relacionan un XML de un layout con una clase Java, de tal manera, que es posible gestionar desde el Activity la vista generada, porque es este elemento el que crea una pantalla con la vista que contiene los campos definidos en el layout, y los eventos que le afectarán. En realidad y aunque se esté definiendo el Activity en esta sección, debido a su naturaleza, un Activity está formado por ambas partes, la gráfica y la lógica.

2.4.2.1. Ciclo de vida de un Activity

Dentro de las Activity será absolutamente necesario desarrollar el método onCreate que instanciará la vista, es decir, hará que exista la pantalla en la aplicación. En ese método y a través de la clase R que identificará el layout XML que represente al Activity y del método ya existente SetContentView se relacionarán ambos partes. Cabe destacar que aunque se haya creado una vista a través de este método, esto no significa que se visualice, sino que simplemente estará disponible para su visualización si se realiza una llamada a la pantalla o vista y es en este punto dónde entra en juego el ciclo de vida de las Activity. Para ello se incluirá en el documento, el gráfico que proporciona la propia documentación oficial y se explicarán cada una de las fases de las que constan:

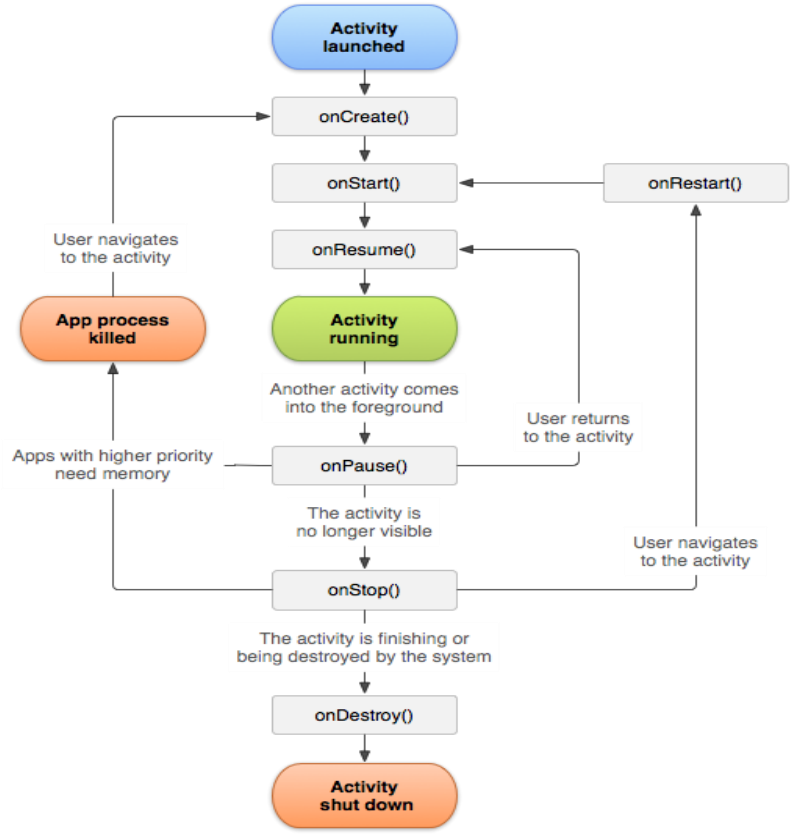

Figure 27: Ciclo de vida de un Activity

- Fase onCreate → se encarga de instanciar la vista e inicializar las variables y la interfaz de usuario
- Fase onStart → hace la vista instanciada visible para el usuario aunque no tiene que estar obligatoriamente en un primer plano, exactamente como ocurre cuando se despliega el menú lateral. En ese caso la vista sigue visible pero por detrás del menú
- Fase onResume → la vista se encuentra en un primer plano. Es decir, que a diferencia de la fase anterior en este punto es la vista la que se encuentra por encima del resto
- Fase onPause → este es un ciclo alternativo, ya que no tiene por qué formar parte de la vida de todas las vistas. La vista se mantiene en un plano inferior y se encuentra pausada hasta que desaparezca la vista que se había quedado en un primer plano de tal manera que al finalizar este paso la vista vuelve a estar delante del resto
- Fase onStop → la vista deja de estar visible. No es que se encuentre por debajo de otra vista y se vea parcialmente, es que no se visualiza
- Fase onRestart → esta fase sólo puede darse cuando una vista se encuentra en el paso onStop y se encarga de volver a hacer visible una vista una vez que ha dejado de ser visible en la fase anterior
- Fase onDestroy → la vista desaparece totalmente, es destruida y por lo tanto no puede seguir utilizándose ni en un primer plano, ni en un segundo ni en ninguno

Otro de los conceptos de la parte "programática" que es necesario aclarar antes de exponer los módulos desarrollados, es el de Fragment. Los fragmentos, como explica la documentación oficial de Android, representan el comportamiento o una parte de la interfaz de usuario en un Activity. Esto quiere decir que los fragmentos serán vistas parciales que podrán incorporarse a una vista de un

Activity para su visualización.

Al ser vistas parciales es posible combinar múltiples fragmentos en una única Activity y también utilizar el mismo fragmento en múltiples Activity, ninguno de los modos está limitado a un único uso, ya que si lo estuviera perdería totalmente la flexibilidad que proporciona. Cada uno de los fragmentos contará con su propio ciclo de vida, que coincidirá con los ciclos de vida que se observaron en la parte de las Activity, pero al estar contenido dentro de un Activity, es importante recordar que los fragmentos no pueden existir por sus propios medios, dependerá directamente del ciclo de vida del Activity en el que se encuentre. Es decir, si el Activity se encuentra pausado, se encontrarán pausados todos y cada uno de los fragmentos que la compongan. Por el contrario, el Activity podrá estar en la fase de start y estar sus fragmentos pausados o estar todos en ejecución. Al final de cuentas ambos se usan para lo mismo y compartirán ciertas características como vistas que son.

2.4.2.2. Persistencia de datos

Además de lo visto con anterioridad, otro punto fundamental que hay que entender y desarrollar en la parte "programática" de Android es la de la persistencia de datos [26]. La persistencia de datos consiste en el almacenamiento de datos de una aplicación en el propio terminal móvil para que esté accesible en accesos posteriores. En otras palabras, los datos y estados se mantienen de una sesión a otra, incluso si se cierra el proceso de la aplicación. Si no existiese la persistencia de datos sería como si la aplicación se instalase de nuevas y el usuario tendría que volver a crear todos los dashboard y widgets cada vez que reiniciase la aplicación. Por ello, es necesario gestionar los datos que se almacenarán en el dispositivo. Android ofrece distintas opciones y alternativas para llevar a cabo estas funciones y se enumerarán todas ellas a continuación.

- La primera de todas es la de preferencias compartidas → esta persistencia de datos consiste en el almacenamiento de pares clave-valor de cualquier tipo de datos primitivos, es decir, que únicamente permite datos "byte, short, int, long, double, float, char, String y boolean", lo cual significa que no es capaz de almacenar objetos
- Almacenamiento interno → guarda los datos en archivos privados de la aplicación. Con privados el documento se refiere a que únicamente serán accesibles por la propia aplicación, ya que Android cuenta por cada desarrollo un área o zona de almacenamiento inaccesible por otras aplicaciones. De hecho, ni siquiera el propio usuario tiene acceso directo a estos datos si no es a través del desarrollo
- Almacenamiento externo → además de ser capaces de almacenar los datos en ficheros privados internos, todo dispositivo Android permite el almacenamiento externo compartido. En este caso, con la palabra externo no se refiere a almacenar los datos en la nube, sino que siempre se almacenarán en el dispositivo, pero a diferencia del caso anterior, se podrán almacenar o en un área accesible por el usuario y por el resto de aplicaciones o en la tarjeta SD de almacenamiento externo. Al contar con accesos ilimitados a este tipo de ficheros, significa que el usuario podrá modificarlos manualmente a su gusto lo cual puede incurrir en incoherencias o incluso en un borrado accidental que haga desaparecer todos los avances (por supuesto siempre se refiere a avances y modificaciones realizadas por el propio usuario). Para poder guardar ficheros en la zona compartida es necesario solicitar el permiso de escritura en el terminal
- Base de datos SQLLite → es un motor de base de datos bastante extendido y una de las principales formas de guardar datos de las propias aplicaciones, ya que proporciona e integra toda la estructura de tablas y las opciones de consultas, creación, modificación y borrado de registros basándose en un subconjunto de la sintaxis del conocido lenguaje SQL, por lo que muchos desarrolladores escogen esta opción de persistencia, ya que en términos de rendimiento es bastante más eficiente que el resto de elecciones
- Conexión de red → consiste en almacenar los datos fuera del dispositivo en un servidor que se encuentre bajo el control del desarrollador. Por supuesto, al ser un acceso a internet se deberían configurar los permisos pertinentes para que el terminal pueda acceder a los datos y se necesitará

un desarrollo paralelo en el servidor para que sea capaz de devolver los datos consultados y para poder escribir en él los datos que se modifiquen o que sean de nueva creación

Sobre cada uno de los tipos de persistencia conviene exponer algunas de sus ventajas, desventajas y curiosidades de cara al desarrollo Android, así que no estaría de más mencionarlos en esta sección para que se sea consciente de ellos.

Aunque SQLLite es una de las opciones preferidas entre los usuarios y desarrolladores Android, la mayoría de ellos no lo utiliza directamente, ya que su desarrollo produce un código difícil de interpretar y hasta cierto punto sucio, haciendo que su modificación posterior sea muy trabajosa y complicada. La cuestión es que a la hora de generar y gestionar la base de datos, es necesario incorporar directamente al código sentencias SQL del tipo:

```
db.execSQL("CREATE TABLE Dashboard (id INTEGER, nombre TEXT)");
```

Es decir, que se entremezclan las sintaxis de ambos lenguajes dando lugar a un desarrollo algo confuso. Por ello y como se ha comentado momentos antes, los desarrolladores prefieren utilizar librerías ORM (Object-Relational Mapping) que les ahorran toda esa sintaxis de más, permitiéndoles utilizar y mantener las ventajas de un sistema relacional de bases de datos. Los sistemas ORM se encargan de transformar el sistema de datos orientado a objetos a un sistema relacional, es decir, que finalmente utilizará SQLLite por debajo, pero el usuario podrá abstraerse de ello. El problema a esto es que existen múltiples librerías que proporcionan un sistema ORM y el desarrollador deberá estudiarlas para saber cuál de todas se ajusta mejor a sus necesidades, ya que no en todos los casos las librerías soportan toda la versatilidad que existe en Java para la definición de las clases y sus objetos, por lo que es posible dar con algún caso en el que no se pueda continuar el desarrollo con la librería escogida debido a que esta no ofrece soporte para las necesidades del usuario.

En el caso de la conexión de red y como se ha comentado anteriormente, no suele ser una de las opciones predilectas por los usuarios, ya que requiere del desarrollo complementario de servicios en la parte del servidor para poder servir los datos a la aplicación cuando esta los solicite y puede suponer una carga extra de trabajo para el desarrollador. De hecho, no en todos los casos se está dispuesto a realizar este trabajo extra e incluso en otros no se cuenta con el tiempo necesario para ello. Lo bueno de este tipo de persistencia de datos es que permite la compartición de la información no únicamente por parte de otras aplicaciones con las que se cuente en el propio dispositivo sino que el acceso directo por parte del usuario o por parte de otro tipo de servicios o aplicaciones, como pueden ser las de escritorio, se simplifican enormemente, por lo que es una opción que siempre tiene que barajarse antes de decidirse por un modo u otro de persistencia de datos. Así que si se cuenta con una aplicación que tiene tanto versión móvil como versión de escritorio, puede ser interesante que ambos compartan la información y se sincronicen para ofrecer un sistema más completo y coherente.

En el caso de las preferencias compartidas al no poder almacenar objetos, suele usarse para datos simples o de configuración, aunque hay desarrolladores que le han dado una vuelta de tuerca a este formato y lo que han hecho es generar sus datos en forma de objetos y luego para poder guardarlos en las preferencias compartidas lo han transformado en un String que luego son capaces de volver a su estado inicial invirtiendo la transformación. La mayoría de las veces lo que suele encontrarse en ese String es un Json y esta transformación se hace porque el acceso a las preferencias en valores de rendimiento son mucho mejores que la lectura de cualquier fichero o el acceso externo a los datos. Debido a ello y a que crear y leer las preferencias es un proceso sencillísimo, es una práctica muy habitual.

Tampoco se debe olvidar la persistencia de datos por ficheros tanto internos como externos y tanto privados como compartidos, ya que ambos son opciones muy flexibles que permiten que se

almacenen los datos en el formato que le sea más cómodo al usuario. Además de eso, el hecho de tener un fichero físico permite la exportación manual de los datos de forma más cómoda para el usuario, ya que el archivo tiene existencia propia dentro del sistema de ficheros. La ventaja principal de esto es poder llevar los datos de un dispositivo a otro o incluso generar backups y evitar así la pérdida de información fundamental. Esto no significa que únicamente este tipo de persistencia permita la exportación de los datos, pero sí que es el que menos desarrollo para ello necesita y por lo tanto es algo a tener en cuenta. A pesar de todas esas ventajas, tiene una desventaja clara y es que la lectura y escritura, es decir, el acceso a estos ficheros penaliza enormemente el rendimiento de la aplicación y por eso, en ocasiones se ha optado por otras de las opciones. Una de las soluciones para estos casos es la de cargar y leer todos los datos directamente al acceder a la aplicación y gestionar el objeto resultante desde el resto del código evitando posteriores accesos a excepción de casos concretos como puede ser el de escritura o modificación de información. Debido a todas estos puntos, la opción de persistencia en ficheros únicamente se suele llevar a cabo para ficheros cortos o para aplicaciones con pocos accesos a disco, ya que en estos casos el rendimiento puede ser igual de bueno que el de la propia base de datos de SQLLite que se pudo ver un par de párrafos antes, mientras que su alternativa natural será la de generar los datos a persistir en el formato que se desee y transformarlo posteriormente a un String que se guardará en el sistema de preferencias compartidas.

2.4.2.3. Gradle & Manifest

"Gradle" es una herramienta de automatización y configuración de la aplicación en la que se puede definir la SDK Android mínima (parámetro minSdkVersion) que se soportará o lo que es lo mismo, cuál es el API y versión mínima de plataforma de Android al que se ofrecerá soporte. Esto significará que la aplicación sólo podrá ser ejecutada en terminales cuya versión del sistema operativo sea igual o superior a la mínima definida en el Gradle. Además, para cualquier proyecto Android determinará qué dispositivos pueden usarlo, también definirá qué funcionalidades de Android se podrán usar para desarrollarlo, es decir, que el equipo de desarrolladores de Android va incorporando nuevas funcionalidades o mejoras de las ya existentes en las nuevas versiones de SDK y que por lo tanto sólo estarán disponibles desde ese punto y no antes, lo que limita en algunos casos cómo deben desarrollarse las aplicaciones, ya que no se cuenta con los últimos avances en el desarrollo de la tecnología Android. De esta manera, cuanta más alta sea la SDK fijada en el fichero Gradle, menos porcentaje de terminales Android podrá usarlo, pero se contará a su vez con una mayor capacidad de desarrollo, así que si se quiere llegar a un gran porcentaje de usuarios sin penalizar el trabajo en términos tecnológicos, se necesitará llegar a un punto intermedio y hacer balance entre ambos puntos para equilibrarlos.

Además de la SDK mínima, otros parámetros configurables del fichero Gradle comprenden el compileSdkVersion que fija la versión en la que se compilará para generar la aplicación. Por lo tanto, cuanto mayor es esta configuración más podrá aprovecharse de las ventajas que ofrecen las versiones más actualizadas, dentro de las cuáles se encuentran arreglos de bugs o fallos.

Por último cabe destacar la existencia de otro parámetro, targetSdkVersion, que se encarga de gestionar la compatibilidad hacia delante, es decir, que está íntimamente relacionado con el valor del minSdkVersion y representa realmente para qué versión está desarrollada la aplicación. De esta explicación, es posible inferir lo siguiente, que es una regla globalmente extendida:

minSdkVersion <= targetSdkVersion <= compileSdkVersion

A su vez, también está recomendado lo siguiente:

minSdkVersion (siendo la más baja que se pueda incorporar) <= targetSdkVersion <= compileSdkVersion (siendo la más actualizada posible)

Esto significa que al desarrollar en esta tecnología se debe intentar soportar el máximo número de dispositivos, con las funcionalidades actualizadas y el máximo número de errores corregidos en la

compilación y generación de la aplicación.

Al intentar soportar un rango de versiones SDK es necesario algún modo de añadir funcionalidades nuevas a versiones anteriores o crear funcionalidades que sean lo más parecido a esas funcionalidades de las SDK nuevas, pero que a su vez sean compatibles con lo que permite hacer la versión de SDK inferior, en otras palabras, sería conseguir funcionalidades equivalentes, y es ahí donde entra la SupportLibrary o librería de soporte o apoyo. Esta librería añadiría una capa de compatibilidad entre las distintas SDKs. Debido a esto y a la definición de los valores SDK en el Gradle, la SupportLibrary deberá ser igual o superior al valor del parámetro de configuración compileSdkVersion, ya que a la hora de compilar el código y generar la APK final con la aplicación deberá utilizar la librería e incorporar esas equivalencias que permitan el uso de la aplicación en múltiples terminales que cuenten con distintos sistemas operativos Android.

Con esta información y basándose en el gráfico que ofrecen los creadores de Android sobre el uso de las versiones del sistema operativo [24] y que se ha incorporado a continuación de este párrafo, es posible deducir el porcentaje de dispositivos Android que hay actualmente en el mercado que podrían utilizar la aplicación.

Version	Codename	API	Distribution
2.3.3 -2.3.7	Gingerbread	10	1.0%
4.0.3 -4.0.4	Ice Cream Sandwich	15	0.8%
4.1.x	Jelly Bean	16	3.2%
4.2.x		17	4.6%
4.3		18	1.3%
4.4	KitKat	19	18.8%
5.0	Lollipop	21	8.7%
5.1		22	23.3%
6.0	Marshmallow	23	31.2%
7.0	Nougat	24	6.6%
7.1		25	0.5%

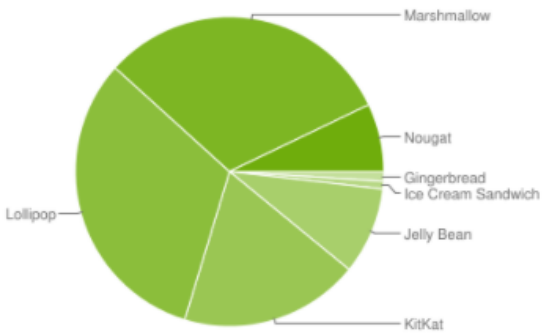

Figure 28: Gráfico del uso de versiones Android

Además de para definir los valores de SDK, que como se ha visto es un punto clave en el desarrollo, en el Gradle será donde se gestione todo lo relacionado con librerías externas a través de un gestor de paquetes Maven.

Para ello es necesario definir el repositorio dónde puede ser encontrada la librería y después añadir la llamada de compilación de la misma como una dependencia del desarrollo. Únicamente con esas dos líneas ya es posible usar la librería en cualquiera de las clases Java, siempre y cuando se importe en ellas las funcionalidades que se van a utilizar.

Además de estos ficheros, es importante recalcar la existencia de otro fichero esencial de configuración, este es el AndroidManifest.xml. Este fichero se encuentra en la raíz de cualquier aplicación Android y contiene información clave como por ejemplo:

- El nombre del paquete que se utiliza en la aplicación, que no es sino un identificador que representará a la aplicación
- Los permisos que la aplicación necesita para ejecutarse. Estos permisos se solicitarán al

usuario a la hora de instalar la aplicación a través de alguno de los portales de descargas de aplicaciones. Esta parte es totalmente necesaria para conseguir acceder a partes de la API que están vetadas a no ser que se tenga un permiso concreto o para interactuar con otras aplicaciones

- Lista las librerías que deben ser enlazadas

Este fichero y todos sus datos se definirán en formato de etiquetado XML por lo que los módulos pueden verse y entenderse con un único vistazo rápido.

3. Diseño y desarrollo

En el desarrollo de la aplicación se ha ido evolucionando un proyecto básico de Android desde una actividad básica hasta una aplicación que permite la visualización tanto textual como gráfica de entornos IoT. Como es de esperar, la versión inicial generada a través de Android Studio, proporcionaba la estructura básica y necesaria para poder ejecutarse en un terminal móvil.

3.1. Actividades

Una vez generada una solución básica en Android Studio, se procedió a desarrollar la única Activity con la que cuenta el proyecto.

3.1.1. Actividad principal

La Activity principal estará representada con el layout activity_main.xml y el content_layout.xml. Mientras que el primer layout carga el contenido del menú, el segundo contendrá un FrameLayout para ir representando en ella los diferentes fragmentos de representación de vistas de navegación. En realidad, este último layout es el único gestionado por una clase Activity, es decir, que no es un fragmento en sí mismo y se utilizará para proporcionar la lógica inicial de la aplicación e instanciará el resto de fragmentos para gestionar el flujo de la aplicación. En otras palabras, esta actividad se encargará de gestionar la visualización de los fragmentos de las distintas secciones.

3.1.2. Menú lateral y navegación

En realidad, y como se ha mencionado anteriormente, este layout forma parte de la vista del layout principal y únicamente utiliza un elemento NavigationView con un listado de opciones que se rellenan a través la clase Java. Además de esto se le añadió una vista simple para la cabecera del menú. De tal manera que sea posible añadir una imagen representativa de la aplicación y un texto con información relevante. Esta vista de la que se está hablando en este apartado, se encuentra en el fichero header_navview.xml.

Lo primero que se debía hacer para crear un menú deslizante era añadir la librería android-support-v4, ya que es a partir de esta versión de librería de soporte desde donde se cuenta con la clase Navigation Drawer. En este caso, fue capaz de saltarse este paso porque al generar el proyecto básico desde Android Studio, añade por defecto la librería appcompat-v7 que a su vez depende de la primera por lo que incluye ambas desde el inicio.

Una vez hecho esto, se sustituía el contenido del activity_main.xml por un DrawerLayout con dos elementos, siendo el primero de ellos una llamada a otro layout que contendrá una vista principal de la aplicación y el segundo de ellos representará el menú de navegación NavigationView. El desarrollo de esta parte se basó en las buenas prácticas de otros desarrolladores Android que recomendaban el uso del MainActivity y su layout_main asociado como contenedor y gestor de la navegación principal, de tal manera que se centralizará en un único fichero y que las vistas de contenido principal estuvieran aisladas de este segmento.

Como con el control NavigationView, únicamente se notificaba que se contaba con un componente menú, era necesario añadir por la parte de “Layout” una vista en la que se definieran las opciones del mismo. Para ello, únicamente era necesario generar un xml de recursos con el listado enumerado de las opciones siguiendo la sintaxis típica de los xml de recursos de tipo menú, ya que anteriormente ya se había asignado ese fichero al propio NavigationView. Cabe destacar que al ser un fichero de recursos estos elementos deben codificarse de manera individual y manual por el usuario. Otra de las cosas que se decidieron en el momento de la inclusión del NavigationView, era que al igual que en la mayoría de aplicaciones Android el menú se deslizaría desde el borde izquierdo, usando la propiedad layout_gravity con valor “start” y que se añadiría una vista extra como cabecera del menú para poder añadir información general de la aplicación cuando el menú se despliegue. Con este desarrollo ya se

podría ver aparecer y desaparecer un menú, pero además de esto, se necesitaría un layout principal para el contenido central de la página.

Ahí es donde entraría en juego el fichero content_layout.xml al que se cargaría desde el main_layout.xml. Ese layout definiría no sólo su tipo, que sería un LinearLayout, sino también lo que contendrá, que en este caso se ha decidido que se traten de Fragments, ya que se definirá cada una de las páginas como un fragmento único que podrá gestionarse y controlarse de manera individual.

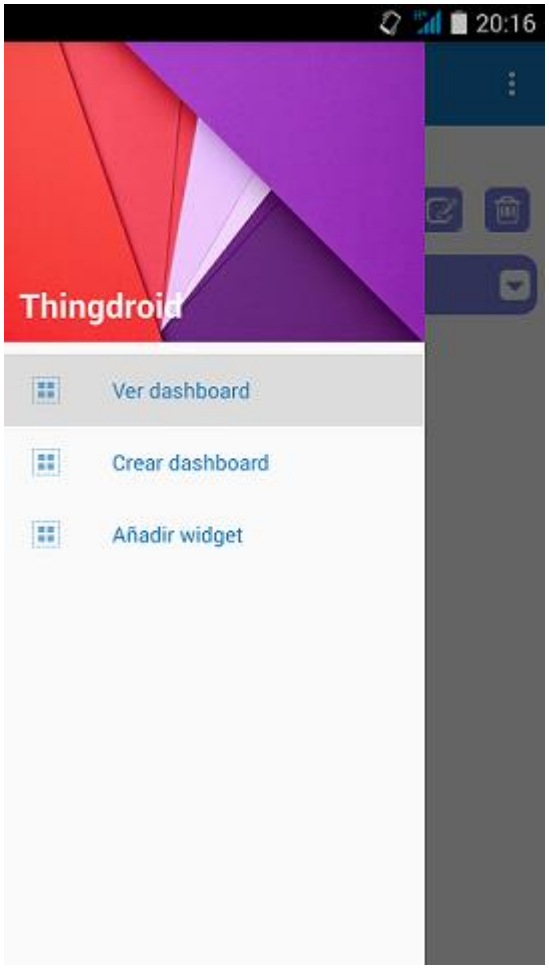

Figure 29: Menú de navegación

Una vez que se contara con la parte visual resuelta, tanto del menú como de los fragmentos de las vistas, se debían gestionar los eventos que se produjesen con cualquier acción relacionada con el menú de navegación y como se ha comentado al inicio será trabajo del Activity principal. Lo primero que se realizó, al igual que se hace para cualquier otra Activity, era encontrar el elemento en la vista para poder utilizarlo desde Java. Justo a continuación de esto, se agregará al elemento un listener de eventos para detectar los click sobre las opciones del menú a través de la función existente en Android setNavigationItemSelectedListener. Con esta función se es capaz de determinar qué opción se había elegido y realizar distintas acciones según la elección. Finalmente para cada una de las opciones se activaría la instanciación de su fragmento correspondiente y la substitución del fragmento actual por el de la vista a la que se pretende navegar. Por supuesto, esta navegación se realiza a través de las funciones que proporciona la clase Fragment que extiende la funcionalidad de las actividades.

Esto no significaba otra cosa sino que era necesario tener un fragmento por cada vista navegable del menú. Debido a ello se crearon varias vistas simples accesibles desde el menú de navegación, quedando finalmente las secciones de "*Ver dashboard*", "*Crear dashboard*" y "*Añadir widget*". Cada uno de estos fragmentos se expondrá en la siguiente sección.

3.2. Fragmentos

A partir de este punto se irán definiendo y explicando todos y cada uno de los layout que se han creado para el proyecto, empezando por los de las secciones navegables del menú.

3.2.1. Fragmento ShowDashboard

El ShowDashboard y su layout fragment_show_dashboard.xml es probablemente el fragmento más importante de todos. Este fragmento se carga por defecto al abrir la aplicación y se encarga de mostrar un listado de widgets pertenecientes a un dashboard en concreto. El dashboard que se mostrará por defecto, será aquel que tenga como propiedad 'primary' igual a 'true', aunque también permitirá a través de un elemento spinner el cargar los contenidos de cualquier otro dashboard que se tenga almacenado en la base de datos de la aplicación.

Como se ha especificado en el resumen de arriba, el desarrollo de este fragmento consistía en gestionar la carga de widgets de un dashboard concreto. Para ello, se debían leer los datos almacenados en el principal y encontrar cuál de todos ellos era el principal. Una vez habiendo

recogido esta información se debían instanciar todos y cada uno de los widgets que estuvieran incluidos en ese dashboard. El procedimiento de carga de widgets era prácticamente igual a lo que se había realizado anteriormente para la carga de este propio fragmento. La diferencia radicaba concretamente en dos conceptos. El primero de los dos, consistía en que en este caso, un fragmento no debía sustituir al anterior, sino que debían mostrarse todos a la vez, uno a continuación del siguiente. El segundo, es que dicha instanciación del fragmento, requería el traspaso de información de un fragmento (el de showDashboard) a otro (el propio del widget).

Para ello se crearon un par de instancias de widget y se investigó cuál era la mejor manera de traspasar los valores de las instancias al layout del fragmento. De primeras, se probó a traspasar cada uno de los atributos independientemente a través de la clase Parcelable con el método putString para los algunos de los valores del modelo de datos, pero enseguida se podía vislumbrar que aunque la información sí que llegaba al layout y era posible mostrarla, tener que hacerlo para cada campo era inmanejable y más si se tenía en cuenta que se debería hacer esto mismo para cada tipo de widget por lo que se decidió implantar una alternativas. La primera opción que se barajó basándose en algunas recomendaciones de otros desarrolladores Android era hacer que la clase del widget serializara el contenido del objeto y luego fuera el fragmento el que lo leyera. El problema que se encontró con esta solución es que sólo era recomendable para objetos pequeños debido a su bajo rendimiento y aunque era cierto que el objeto en cuestión era bastante simple, dada la naturaleza de otros widgets era más que probable que no todos lo fueran y que por lo tanto la adaptación acarrease código duplicado y mal gestionado.

Debido a que se quería mantener una cierta homogeneidad entre las clases y especialmente entre los desarrollos de los widgets se decidió apostar por una segunda opción que es la que finalmente se ha adoptado en el proyecto. Esta opción era la de hacer que las clases de los widgets implementaran la clase Java Parcelable y con esto sería posible para el usuario, traspasar la información de un objeto completo a un fragmento. Para ello, se debía conseguir que la clase del widget implementara la interfaz del Parcelable para así poder contar con las funcionalidades que ofrece. Debido a este cambio era necesario crear algunos métodos específicos como el writeToParcel en el que se volcarían los atributos del widget dentro de un objeto de tipo Parcel, un constructor que generara la instancia a través de un objeto Parcelable y otro método llamado readParcel que realizará el proceso inverso para poder recuperar un objeto de widget desde un Parcel y por último para los casos cuyos uno de los atributos sea una lista tipada, será necesario generar un objeto Parcelable.Creator que no es más que un objeto que genera otros objetos de un tipo personalizado, es decir fuera de los definidos en Java, a partir de un objeto Parcel. En otras palabras, sería complementario al readParcel. Un dato relativo al tema de la lectura y escritura de Parcelable y que es importante de destacar es que se debe mantener el orden de los campos tanto en la lectura como en la escritura es indispensable, ya que sino no es capaz de saber qué valor corresponde a cada campo en términos de propiedades de una clase. De esta manera mejora enormemente la organización del traspaso de información de este componente a otro Fragment.

Referente a este tema, se ha estudiado documentación complementaria acerca de que es completamente posible realizar este mismo traspaso de la información a través de un setter, pero que haciéndolo con ese elemento, se perdía la funcionalidad de restaurar el estado del fragmento, por lo que los datos que se modificasen en la aplicación podrían perderse si se saliese del foco de la misma para utilizar otra aplicación.

Esos datos que a partir de este punto ya eran accesibles desde la clase propia del widget, utilizando para ello la funcionalidad del Parcelable y de los bundles, debían mapearse con sus controles en la vista. Para ello, se generaba un nuevo objeto de widget accediendo a los datos a través del método getParcelable. Con los datos ya en la clase, se recuperaban los controles afectados uno a uno y se modificaba la vista con ellos. Por ejemplo, para los elementos de tipo TextView, el asignar su valor era tan simple como usar el método setText con el parámetro de cadena textual recogida del susodicho objeto con un get de los definidos en el modelo de datos, pero esto dependería de los controles con lo

que cuenta el widget que además varían según el tipo.

Esas modificaciones debían realizarse para cada tipo de widget, de tal manera que se pudiera continuar con el desarrollo de la clase ShowDashboard incluyendo un bundle por cada instancia de widget y utilizando el método PutParcelable. Con esto ya se tendría preparada toda la información para que el fragmento la recibiera, pero para llevar eso a cabo, primero debía existir el fragmento. De hecho, y como se ha comentado con anterioridad, se creará un fragmento por widget y por último, estos fragmentos se añadirían a la vista del ShowDashboard uno a continuación del otro.

Al poner un par de estos widgets se descubrió que si se incluía un número suficiente de ellos no se era capaz de visualizarlos todos en la pantalla porque se salían por la zona inferior, así que se añadió a la vista parcial de fragment_show_dashboard.xml un elemento ScrollView que englobara el elemento layout en el que se iban agregando los fragmentos de widgets. En este caso, sólo se debía ajustar el scroll al contenido en altura, que era donde lo superaba. Para el ancho se forzaba a que se ajustara al dispositivo.

Esta funcionalidad del scroll es posible probarla incluyendo en el layout distintas alturas para los fragmentos, es decir, forzando unas alturas fijas para cada uno de los elementos o instanciando múltiples elementos hasta que salgan de la pantalla.

Para agregar todos y cada uno de los fragmentos se optimizó a través de una función independiente que gestionaría la creación de los distintos fragmentos basándose en la clase del objeto que era posible recuperar y usando un switch para los diferentes widgets. Por lo tanto, por cada valor de la clase, tendría una instanciación de fragmento distinta. Además, la instanciación de los widgets se habría generado de manera independiente y sin relacionarlos con el objeto Dashboard más allá del hecho de que se habían obtenido del listado.

Como se ha comentado en el resumen inicial, este fragmento gestionaba no sólo la visualización y carga de un listado de widgets perteneciente a un dashboard, sino que lo lógico era poder seleccionar entre distintos Dashboard y así mostrar distintos widgets con cada uno de ellos. Para poder llevar esto a cabo, era necesaria la creación de un Spinner, que en otras palabras se trata de un elemento de control Android que genera un componente visual de lista desplegable y que cuenta en su haber con funciones de gestión de selección de opción, etc.

Para ello, se añadiría el elemento gráfico en el fichero fragment_show_dashboard.xml y se empezaría a cargar en el mismo un listado de valores realizándolo con la funcionalidad ArrayAdapter que relacionaba el elemento gráfico con los valores de la lista dentro de un Activity concreta. Una vez visualizada la lista se debían gestionar los eventos touch, o click si se prefiere este término, de selección de opción en el Spinner con el método ya existente setOnItemSelectedListener. En este método podría recuperarse el valor de la opción seleccionada y con ellos se podría saber qué Dashboard se debería visualizar.

Figure 30: Seleción de Dashboard con Spinner primera versión

Sabiendo cuál es el Dashboard que mostrar ya se podían añadir los fragmentos que lo componían, así que simplemente se iniciaba la transición de los mismos, pero se ha observado que en estos casos cómo la función del FragmentManager únicamente añadía fragmentos sin eliminar los anteriores se contaba con un listado incremental y se debían borrar los anteriores antes de añadir los nuevos. Al igual que el FragmentManager contaba con una función add que añadía fragmentos, también cuenta con otra, remove, que los elimina, pero es necesario que se cuente con un parámetro, que será la instancia del fragmento a desechar. Para poder controlar los widgets a borrar, se debe saber con anterioridad cuáles de ellos estaban activos, así que este problema se debía gestionar a través de una lista de fragmentos que se actualizaba cada vez que se cargase un Dashboard, primero eliminando los activos y luego, añadiendo los nuevos a mostrar tanto en la transición como en la lista de activos.

El problema que se vió en este punto es que al seleccionar un Dashboard u otro, mostraba correctamente su contenido, pero no en el orden adecuado, es decir, no mantenía el orden de inclusión del widget en el listado. Sin embargo, no era que lo ordenara a la inversa o algo similar, es que cada vez los ordenaba de un modo distinto, es decir, que seguía un orden totalmente aleatorio con cada selección. Investigando sobre este problema se ha descubierto que es un bug reportado a Android desde hace bastante tiempo, pero que de momento no tiene solución por parte del equipo de desarrolladores. Para poder conseguir el efecto que se deseaba, o en otras palabras, para poder mostrar siempre los fragmentos en el orden en el que se incluyen en la transición, se debían instanciar transiciones nuevas cada vez que se añadieran o se eliminaran fragmentos en la vista. Esto hace que el código sea algo redundante en esta parte al utilizar múltiples instancias, pero según la documentación y comentarios al respecto es la única opción disponible a día de hoy. Con esta modificación era posible observar que ya funcionaba tal cual se esperaba en un inicio y que se podía a proceder a preseleccionar el dashboard principal al acceder al fragmento.

Además, la propiedad de dashboard principal se creó para que únicamente uno de todos los dashboards con los que se contaban en la aplicación pudiera serlo en el mismo momento en el tiempo y que al acceder a la vista "*Ver dashboard"*, fuera este y no otro el que se visualizara de inicio.

Esta última parte del desarrollo del fragmento ShowDashboard, aunque no era realmente necesaria, se realizó para mejorar el UX o usabilidad de la aplicación y para darle sentido a la propiedad de 'principal' que tienen todos los dashboard y que se han estado modificando y actualizando constantemente.

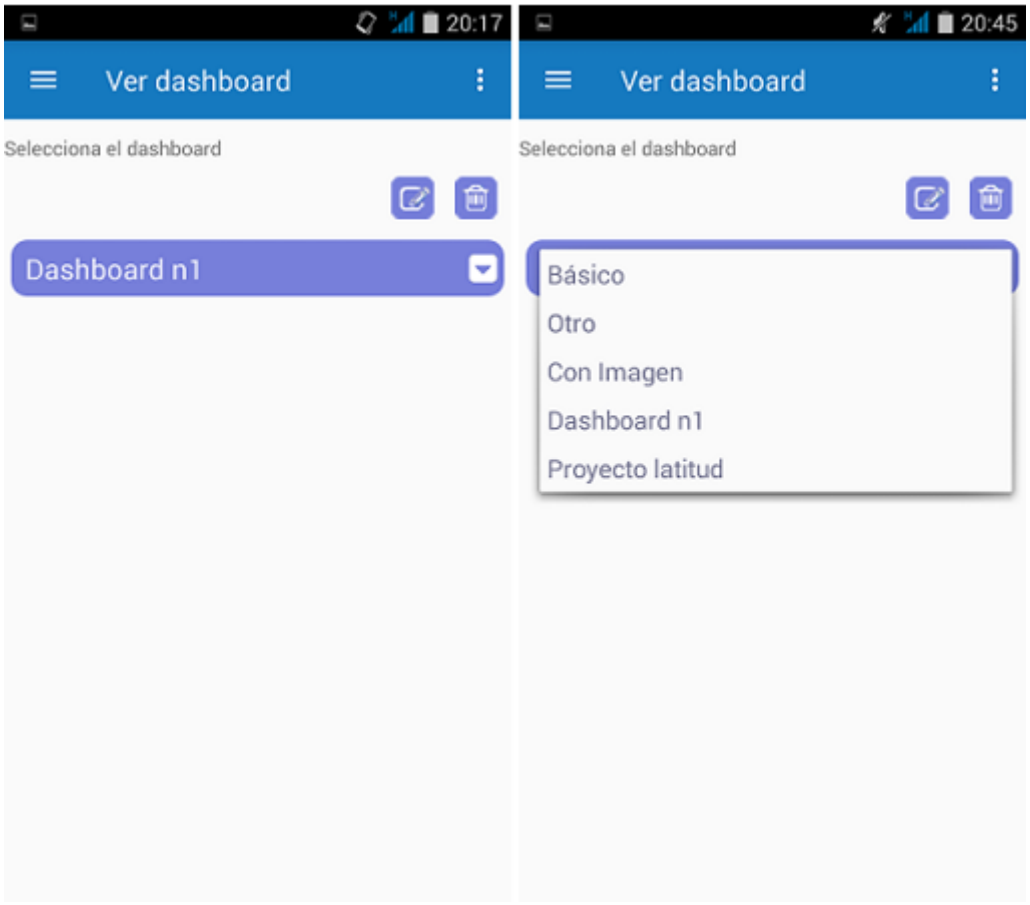

Figure 31: Fragmento ShowDashboard

3.2.2. Fragmento CreateDashboard

Este fragmento, como su nombre indica, contendrá la vista de formulario para la creación de un nuevo dashboard. En ella se mostrarán el campo obligatorio de 'nombre' en un elemento de entrada textual, es decir, en un EditText, que representará al dashboard en la base de datos de la aplicación y el campo 'primario', que se mostrará con un Checkbox, para conocer si este dashboard debe ser considerado el principal y por lo tanto debe ser el mostrado por defecto en la vista ShowDashboard. Además de esto, el botón que se ha incorporado en la vista activará todas las validaciones de los campos antes de que se almacene el nuevo objeto y realizará la llamada a las funciones que se encargan de la lógica del guardado de dashboards.

Una vez se haya procedido a pulsar el botón ‘Crear Dashboard’ se entrará en la lógica del fragmento. Lo primero que se revisará es si el campo ‘Nombre’ se ha rellenado o no, ya que como ese dato representará al propio dashboard de cara al usuario final, es obligatoria su existencia. Si se encuentra vacío en ese momento, no ocurrirá nada, pero en caso contrario, revisará si la lista de dashboards de la aplicación está vacía o no. Si lo está, incluirá este nuevo dashboard, asegurándose de que es el principal. Si no está vacío, será el momento de recorrer el listado de dashboards para confirmar si el nombre introducido es nuevo o si ya existe. Si ya existía el mismo nombre, saldría un cuadro con un mensaje en el que avisaría de este hecho al usuario y el dashboard no se almacenaría. Si no coincidiera con ninguno, entonces se revisaría la opción de marcado CheckBox para el atributo o propiedad ‘primmary’. Si no estuviera marcado, se almacenaría el dashboard tal cual venía en el formulario, si por el contrario el usuario lo hubiera marcado, se le mostraría al usuario un mensaje de que ya existe un dashboard principal y le consultaría para saber si el usuario quiere sustituir el principal por el nuevo introducido en el formulario.

Para ello, solo se requeriría modificar la propiedad ‘primmary’ del actual dashboard principal para

que tome valor ‘false’ y justo a continuación cambiar esa misma propiedad del nuevo dashboard instanciado y almacenarlos justo a continuación. En el caso de haber pulsado el botón de ‘No’ solo se modificaría la propiedad ‘primmary’ del nuevo dashboard para que su valor antes de almacenarlo.

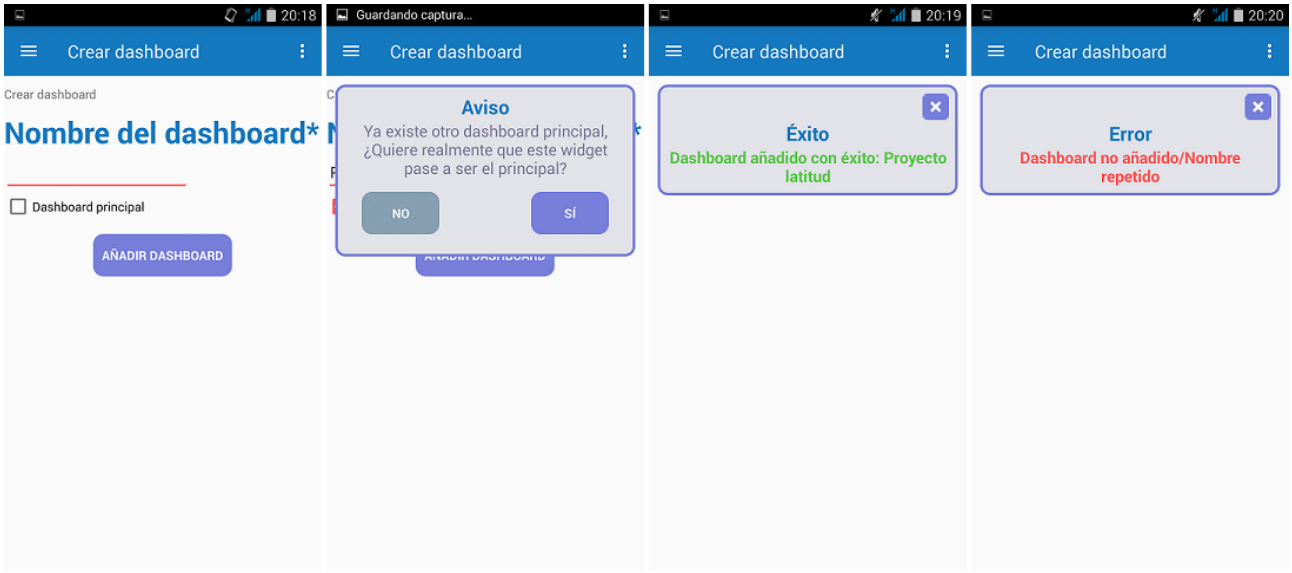

Figure 32: Fragmento CreateDashboard

3.2.3. Fragmento AddWidget

Esta vista se encarga de cargar los distintos tipos de formularios para la creación todos los tipos de widget, trabajo que hará a través de un Spinner que reemplazará un fragmento con otro según el valor que haya sido seleccionado. Las opciones del spinner estarán por lo tanto limitadas a los tipos de widgets. En otras palabras es una vista que se usa mayormente para contener otros fragmentos al igual que ocurre con el fragmento ShowDashboard o con la actividad principal. Para ello contará con el fichero de layout fragment_add_widget.xml.

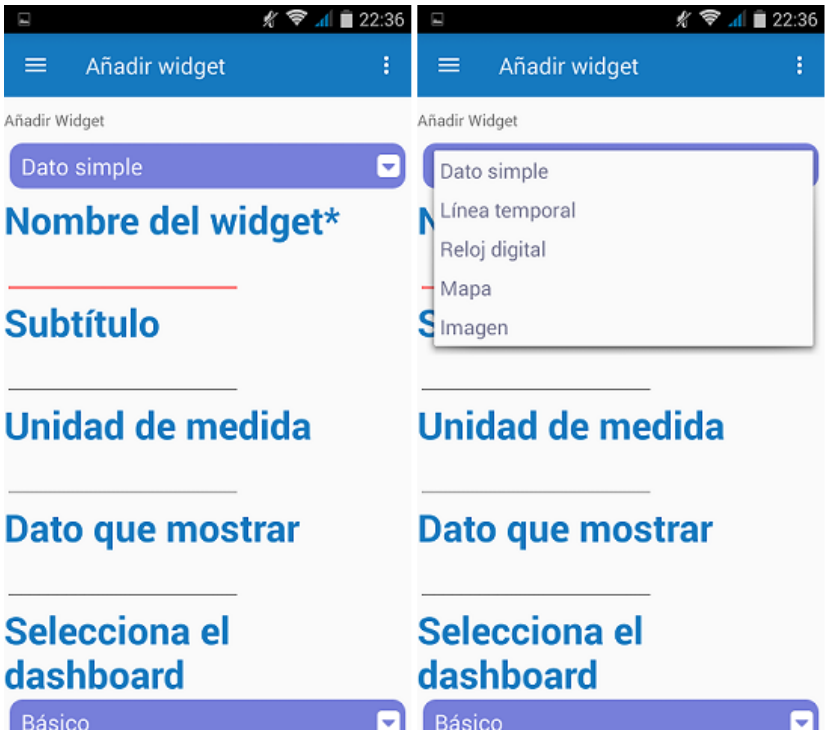

Figure 33: Fragmento AddWidget

3.2.4. Fragmento AddDigitalClockWidget

Figure 34: Fragmento AddDigitalClockWidget

Este fragmento representará el formulario de creación de un widget de tipo reloj en el layout fragment_add_digital_clock_widget.xml. Los únicos valores contribuibles por el usuario serán:

- Nombre
- Subtítulo
- Selección de dashboard

Tanto en este caso como en el resto de fragmentos de creación de widgets, la acción de adición de un widget a un dashboard se realizará cuando se pulse el botón "Añadir widget".

3.2.5. Fragmento AddSingleDataWidget

Figure 35: Fragmento AddSingleDataWidget

Este elemento utilizará el layout fragment_add_single_data_widget.xml que mostrará el formulario para la generación de widgets de tipo valor único. Como puede observarse en la aplicación, este formulario es mucho más completo que el anterior, los valores contribuibles por el usuario serán:

- Nombre
- Subtítulo
- Unidad de medida → mostrará la unidad en la que se medirán los valores recogidos del endpoint
- Dato que mostrar → representará el dato del endpoint que se consultará desde el widget
- Selección de dashboard

3.2.6. Fragmento AddLineChartWidget

Figure 36: Fragmento AddLineChartWidget

Este layout es bastante parecido al de AddSingleDataWidget porque en realidad ambos muestran los mismo datos, siendo la única diferencia que uno de ellos sólo muestra un único valor, concretamente el último, y el otro muestra un listado de ellos que podrá ser configurable por el usuario.

Los valores contribuibles por el usuario serán:

- Nombre
- Subtítulo
- Número de valores → se utilizará para recoger el número de registros que el usuario quiera visualizar en la gráfica
- Dato que mostrar → representará el dato del endpoint que se consultará desde el widget
- Valor del eje Y → mostrará la unidad en la que se medirán los valores recogidos del endpoint y que se visualizará en el eje Y de la gráfica resultante
- Selección de dashboard

Este layout se encontrará definido en el fichero fragment_add_line_chart_widget.xml.

3.2.7. Fragmento AddImageWidget

Figure 37: Fragmento AddImageWidget

Al igual que los anteriores layout de formulario, contará con los campos título y subtítulo, pero además de esto, tendrá un set de botones. El primero de ellos se encargará de abrir la aplicación de galería de imágenes para que el usuario sea capaz de seleccionar una, que se mostrará justo encima de la botonera. Por el contrario, el otro botón servirá para eliminar la misma y que no quede constancia de ella en el almacenaje del widget en la base de datos.

Al pulsar el botón de 'Selecciona una imagen' se generará un elemento Intent que restrinja el tipo de archivos que se permitirán seleccionar, en este caso imágenes, el número de documentos que se permitirán seleccionar, 1 para este caso, y el tipo de acción que se debe llevar a cabo cuando se lance el Activity desde la función, que será:

Intent.ACTION_GET_CONTENT

Está acción levantará la aplicación de galería de imágenes y permitirá elegir una de ellas. Una vez seleccionada, se podrá ver el resultado de la consulta a los documentos desde la respuesta de la misma. Ahí se podrá obtener la URI de la imagen para poder almacenarla y se cargará el contenido del fichero binario a través de un objeto

Bitmap dentro de un elemento de control Android ImageView y en ese momento se haría visible para el usuario.

Los valores contribuibles por el usuario serán:

- Nombre
- Subtítulo
- Selección de imagen → abrirá la galería de imágenes para seleccionar una imagen de las que se encuentran almacenadas en el terminal
- Selección de dashboard

  Este layout se encontrará definido en el fichero fragment_add_line_chart_widget.xml.

3.2.8. Fragmento AddMapsWidget

El último fragmento de adición, será un formulario que además de los campos habituales incluirá uno de latitud y longitud. Si alguno de ellos no es contribuido por el usuario entonces se pondrá por defecto en esa coordenada el valor 0 para que siempre se visualice un mapa aunque el usuario no haya seleccionado ninguna posición cocreta. Los valores contribuibles por el usuario serán:

- Nombre
- Subtítulo
- Latitud
- Longitud
- Selección de dashboard

Por supuesto, los valores por defecto que se introducen en las coordenadas se hará una vez se haya pulsado el botón de "Añadir widget" y el valor no se visualizará en el campo de edición de texto correspondiente.

Figure 38: Fragmento AddMapsWidget

3.2.9. Mejoras en los fragmentos de creación

Para poder mantener una aplicación lo más intuitiva posible, se añadió un pequeño desarrollo complementario para los formularios de creación de dashboards/widgets. Este desarrollo consistía en que en el momento de la generación de estos datos, no sólo se mostrase un mensaje de éxito para que el usuario fuera consciente de que la acción se había realizado con éxito, sino que también limpiaba el contenido de los controles EditText y CheckBox que lo forman. De esta manera el usuario tiene la

certeza de que el cambio se ha logrado, ya que anteriormente podía ser un poco confuso el poder asegurar si una nueva instancia había sido creada o no.

3.2.9.1. Gestión de notificación de mensajes y control de errores en los fragmentos

Para mantener al usuario al tanto de las acciones que se llevan a cabo en la aplicación en relación a los fragmentos de creación de widgets, se han añadido distintos mensajes de aviso.

El primero de todos los mensajes es el que puede ver cuando se inicia la aplicación y no se cuentan con datos almacenados en el terminal móvil, ya que se debe avisar al usuario de que no existe ningún dashboard y por lo tanto no puede mostrar ningún tipo de información. Como se puede deducir de todo el desarrollo de la aplicación, el elemento clave del proyecto son los dashboard, ya que sin ellos no se tienen ni datos de los widgets ni ninguna información que proporcionar al usuario.

Además de mensajes de aviso, también se han añadido recuadros con mensajes para notificar al usuario de que una acción de adición se había o no realizado con éxito cambiando los valores de los texto según ese resultado de corrección. En el caso de la inclusión de widgets dentro de dashboard, únicamente saldría el mensaje cuando realmente se ha añadido el widget, ofreciendo así la información de qué widget (a través de su nombre y en caso de no contar con uno, aparecerá en blanco), se ha incorporado en qué dashboard. De esta manera el usuario cuenta siempre con información adicional que le permitirá seguir perfectamente el flujo de la aplicación.

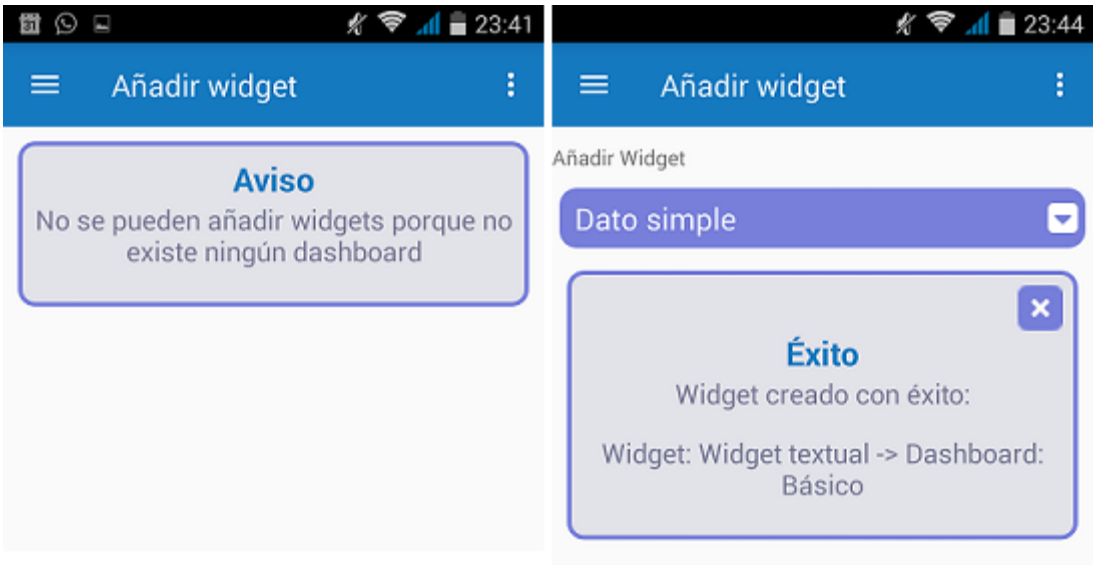

Figure 39: Mensajes de aviso

3.2.10. Fragmento Settings

Para poder proporcionarle más flexibilidad a la aplicación se decidió exteriorizar el uso y gestión del endpoint. Un endpoint es una dirección de acceso a un servicio web que proporciona una serie de datos según una serie de parámetros de entrada. Dichos parámetros podrán ser modificados por el usuario para poder obtener distintas respuestas que se ajusten más correctamente a lo que se espera de él. En profundidad, un servicio web está compuesto por una serie de métodos y funciones accesibles desde el exterior que pueden ser invocados en cualquier momento siempre que se cumplan las condiciones de la llamada, como por ejemplo incluir una serie de parámetros, etc.

El endpoint será por lo tanto la URL de la que se obtendrán los datos externos a la aplicación, o lo que es lo mismo, los datos del proyecto IoT. De esta manera se contará con un proyecto IoT que

contará con su desarrollo propio en el que proveerá esos datos que se quieren visualizar desde la aplicación. Como se ha visto anteriormente, en la actualidad se cuetan con dos tipos de widgets distintos que obtienen sus datos del exterior, el widget de línea temporal y el widget de ato individual. Ambos widgets obtendrán sus datos a partir del mismo endpoint, pero mientras que uno se parametrizará para obtener un único valor, el otro lo hará sobre un conjunto de ellos. La parametrización no sólo del número de valores, sino del campo que se quiere visualizar en el dashboard se hará realmente sobre el propio widget en su formulario de creación, si el campo existe dentro de la respuesta del endpoint, entonces devolverá los valores a través de una llamada asíncrona con la clase *AsyncTask* propia de Android, si por el contrario el campo no existiera, lo que se hará desde el código será rellenar todos los datos (el individual o el listado de la línea temporal) con el valor float 0.0, para representarlo.

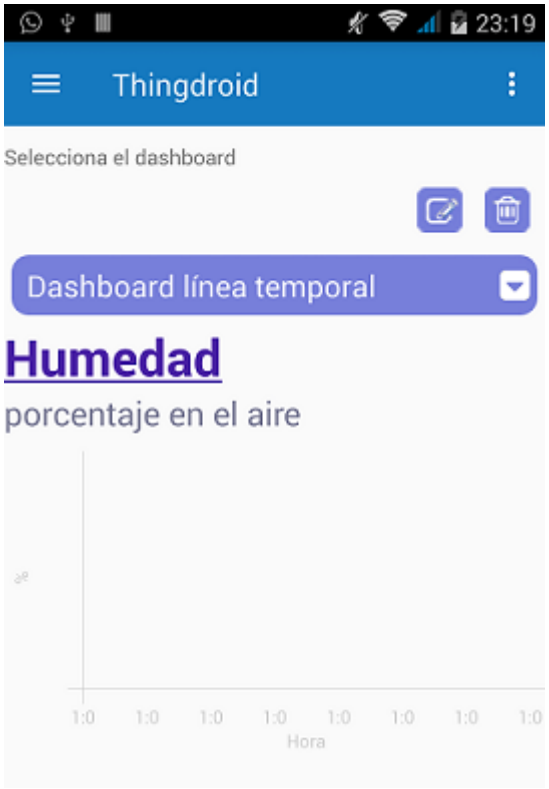

Figure 40: Ejemplo de campo de endpoint inexistente

Por supuesto, se ha definido el acceso a los datos con formato Json por lo que cualquier endpoint que se almacene tendrá que tener formato Json y además, mantener una estructura similar a la que tiene el endpoint de ejemplo, ya que si no cuenta con estas premisas al acceder a alguno de los niveles del elemento Json, es más que probable que devuelva algún tipo de fallo de lectura de Json.

El formato y estructura a la que se refiere el documento es el siguiente:

```
[
{
"ts":1503442569267.0,
"val":{
"device":"C378F",
"humidity":30.080555,
"latitude":41.0,
"longitude":-4.0,
"rssi":-130.0,
"snr":20.68,
"station":"1D62",
"temperature":34.963795
}
}
]
```

Como se puede ver, cada entrada de un registro deberá contar con un timestamp, que refleja la fecha y hora de la toma de la muestra de datos, y de una instancia de los campos que se puede recoger con la medición del proyecto IoT.

Antes de permitir el guardado del endpoint, este se mantenía fijo en la aplicación, a excepción de la parametrización que se ha comentado en los anteriores fragmentos que se podía encontrar en los propios widgets, por lo que la aplicación siempre tomaba los datos del proyecto IoT que se utilizaba de ejemplo. Debido a ello, se podía confirmar sin ninguna duda que era completamente necesario incluir esta parte en el desarrollo de la aplicación. Este dato del endpoint se guardará al igual que el resto de datos en el dispositivo móvil. La variable que se ha utilizado para su almacenaje se llama "*AppDataBaseEndpoint*" y será leída por las funciones de obtención de datos asíncronas que se encarguen de consultar al servicio web.

Además de lo visto anteriormente, cabe destacar que en lugar de generar una nueva sección en el menú principal, se ha creado directamente el cambio de fragmento de 'Ajustes' o 'Settings' en el Toolbar superior, ya que se ha considerado que la existencia del endpoint y de esta sección debe ser un desarrollo horizontal de la aplicación.

Independientemente de lo visto en esta parte, no es para nada necesario contar con un endpoint almacenado, ya que la aplicación cuenta con otros widgets que no dependen de datos externos como el del reloj digital. Con esto lo que se quiere decir es, que aunque la aplicación ofrezca una mayor gama de opciones cuando se tiene endpoint almacenado, no es indispensable, ya que aun sin él, es posible realizar gran parte de las acciones y aprovechar al máximo la funcionalidad que ofrece en su conjunto.

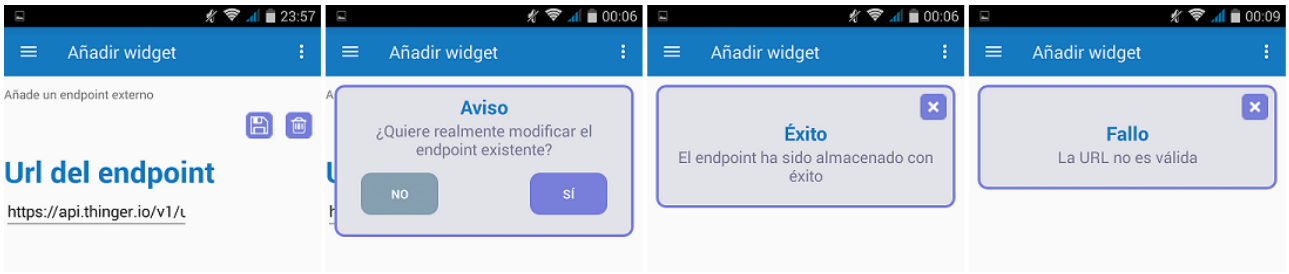

Figure 41: Fragmento Settings

3.3. Widgets

En cuanto a los widgets desarrollados para la aplicación, se enumerarán en esta sección para que los módulos queden perfectamente definidos.

Todos los widgets contarán con su propio fragmento y archivo de layout que se encargarán de hacer visibles los datos almacenados en el dispositivo con su vista correspondiente.

Como era absolutamente necesario contar con una clase "Dashboard" y con una clase por cada uno de los tipos de widgets, al definir la clase "Dashboard" se definieron sus atributos entre los que se podía encontrar un listado de widgets, ya que se dedujo que a cada dashboard debería ser posible añadirle distintos widgets. De hecho, también se definió en este mismo punto que cada widget debería contar con una clase Java que gestionase los objetos correspondientes del tipo de widget o en otras palabras, que definiera el modelo de datos, un layout propio para mostrar los valores del objeto y una clase que gestionase la funcionalidad que compondrá el fragmento. Así que se fueron creando los widgets y definiendo sus modelos de datos..

A la hora de añadir el widget que se había creado, dentro del atributo del listado de widgets del dashboard, se observó que este listado debería permitir múltiples tipos de objetos, tantos como tipos de widget se definieran. Por ello, se diseñó y creó una interfaz que los representara a todos y así el listado se definiría sobre la interfaz.

3.3.1. Fragmento SingleDataWidget o widget de dato individual

Este widget fue el primero que se diseñó para la aplicación. Su funcionalidad consiste en mostrar el valor del último registro recogido en el endpoint sobre un campo específico. Además de esos datos, incorporará una serie de textos informativos para que el usuario pueda leerlos en caso de no acordarse de qué representa el widget que se había generado.

Figure 42: Fragmento SingleDataWidget

Lo primero que realiza el widget de dato individual, es recibir los datos que obtiene del Parcelable que se generaba desde el fragmento ShowDashboard. Con ese Parcelable se podrá generar una instancia del widget para que se adapte a su modelo de datos y con ello, poder obtenerlos de manera individual. Según las propiedades que se encuentren rellenas o no, se mostrarán los campos en el layout correspondiente, fragment_widget_single_data.xml. De hecho, esos mismos controles verán su propiedad de 'visibility' modificada al valor 'gone' para que no ocupen espacio. Si esto no se realizara, el control de Android dejaría el hueco en blanco. Además de mostrar los valores obtenidos del Parcelable tal cual se leen, en el caso del valor obtenido directamente del endpoint, lo mostrará de manera textual, a diferencia de otros widgets que se explicarán a continuación, en el caso de que exista en el Json recogido del endpoint. Para ello, conectará con internet a través de una función Asynctask que realizará la petición al servicio web.

El modelo de datos del widget cuenta con las siguientes propiedades:

- Id → campo textual aleatorio generado como UID. Será un identificador del widget que se usará para localizarlo en un listado de widgets. Su valor en la plataforma será único
- Nombre → nombre del widget
- Subtítulo → subtítulo o descripción textual informativa del widget
- Unidad de medida → unidad en la que se medirán los valores recogidos del endpoint
- Dato que mostrar → dato del endpoint que se consultará desde el widget

3.3.2. Fragmento DigitalClockWidget o del widget de tipo reloj

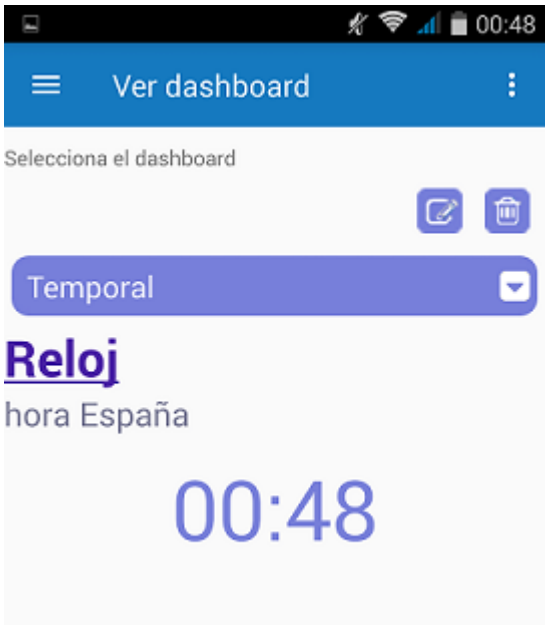

Figure 43: Fragmento DigitalClockWidget

Es una vista que contiene la visualización del widget del tipo Reloj. Este widget muestra el campo textual título y el campo de fecha que representará el valor de la hora leyéndolo del propio terminal móvil.

El widget de tipo reloj estará representado en el desarrollo por la clase DigitalClockWidget. Para este widget, se realizó de inicio la definición de su modelo de datos, sus funciones getter y setter y toda la gestión de los Parcelable porque como ya se había visto desde la creación del primer widget se debía gestionar y hacer posible la transmisión de información entre las Activity. Luego se generaría la clase DigitalClockWidgetPage para gestionar los objetos Parcelable que se reciban de la vista ShowDashboard y por último, haría falta el layout en el que no sólo se representarían los valores de cada uno de los campos del modelo, sino que también se utilizaría un nuevo control, el "DigitalClock" que es

capaz de mostrar la hora actual accediendo a la hora del terminal móvil del usuario, sin tener que realizar ninguna acción o esfuerzo extra. Cabe destacar que en Android es posible añadir tanto un reloj analógico como un reloj digital, pero en este caso se ha elegido el digital para seguir el estilo de la versión de escritorio.

Además, en la imagen que se ha incorporado se puede observar que la hora del widget coincide con la del dispositivo móvil.

El modelo de datos del widget cuenta con las siguientes propiedades:

- Id → campo textual aleatorio generado como UID. Será un identificador del widget que se usará para localizarlo en un listado de widgets. Su valor en la plataforma será único
- Nombre → nombre del widget
- Subtítulo → subtítulo o descripción textual informativa del widget

3.3.3. Fragmento LineChartWidget o widget de línea temporal

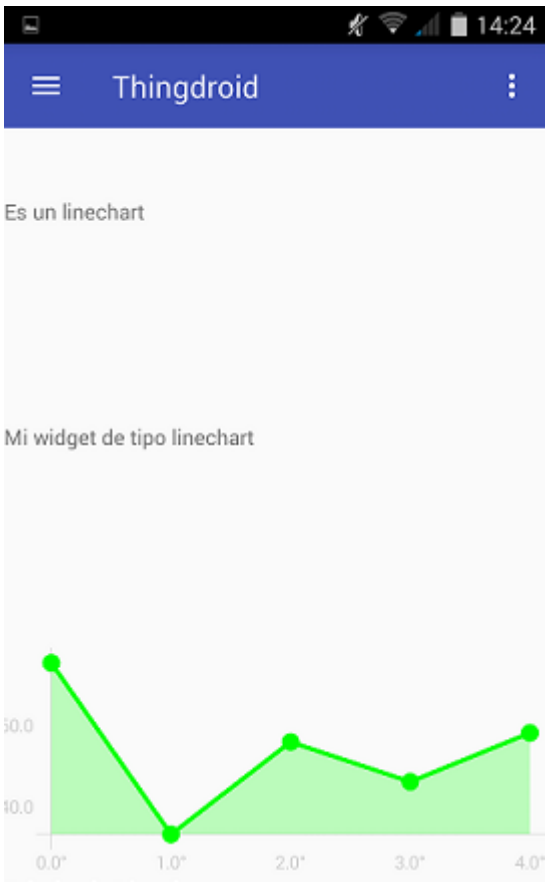

Figure 44: Fragmento LineChartWidget primera versión

Es una vista que contiene la visualización del widget del tipo gráfico de línea temporal. Este widget muestra los campos textuales título, subtítulo, ejes X e Y, y un campo que representa un listado de valores recogidos del dispositivo IoT en una franja temporal concreta. Este widget siempre contará con unidades temporales, en este caso horas, como valores del eje X, coincidiendo con el timestamp o dato temporal de cada registro en el endpoint, de ahí que se denomine línea temporal, mientras que el eje Y sólo mostrará la leyenda de la unidad de medida si el objeto que representa el fragmento tiene la propiedad asignada y los valores numéricos se calcularán tal cual está explicado a continuación.

En este paso, se decidió incorporar un widget algo más complejo a la aplicación. Para poder desarrollar este componente, antes era necesario que se incluyera e integrara la librería que se había escogido para la creación de las gráficas durante el estudio del arte. La librería Java escogida fue "HelloCharts" y se agregó su referencia al archivo Gradle a nivel de aplicación, es decir, de módulo. Una vez agregada la referencia, la cual la se obtuvo de la página oficial de la librería en Github, se sincronizó el proyecto para que pudiera ser usada en los siguientes pasos.

Una vez con la librería disponible y habilitada se pasó como siempre a crear una clase Java que contuviera el modelo de datos y definiera el objeto de tipo LineChartWidget. A priori se contaría siempre con los atributos de título y subtítulo, como era habitual en los anteriores widgets, pero en este caso el campo en el que almacenaría los valores sería un array de valores float, ya que un gráfico de línea temporal se compone de múltiples valores que se van recogiendo en un rango de tiempo determinado y lo que se quiere obtener y estudiar es su evolución en el tiempo. Por último se contaría con los valores de los ejes del gráfico que sería el campo equivalente a las unidades de medida que se podía observar en el widgets de dato individual.

Ya con el objeto y el modelo de datos definido se pasó a crear su fragmento y el layout

correspondiente. Por supuesto, también fue necesario añadir la instanciación de un objeto de tipo LineChartWidget dentro de de un dashboard en el fragmento ShowDashboard.

Lo primero que se hizo en el fragmento fue pintar el resto de los valores de los atributos utilizando los Parcelable igual que lo se realizó en anteriores ocasiones. Esos campos tendrían su correspondiente control dentro de la vista parcial fragment_widget_line_chart.xml, así que lo único que faltaba era crear la gráfica. Por un lado se debía añadir el control específico de la librería en la vista para así luego poder ver los valores de manera gráfica y por el otro lado, es decir, desde el código Java, se necesitaban obtener los valores de los registros del proyecto IoT. Luego esos valores debían ser integrados con las funciones propias de la librería comenzando con los PointValues que representarían los valores en un gráfico de tipo LineChart y agregando ese listado de datos a una línea para conseguir que estén relacionados entre sí. Por último, se reliacionaba esa línea con el propio gráfico. La necesidad de relacionar la línea con el gráfico y de tener ese paso intermedio radica en el hecho de que un gráfico de línea temporal en la librería "HelloCharts" puede contar con múltiples líneas, para así poder realizar una comparación más sencilla y rápida de la evolución de dos sistemas distintos.

Con los datos ya obtenidos e integrados en un objeto LineChartData simplemente se tenía que llamar a la función de la librería setLineChartData, a la cual es posible llamar en cualquier momento que se tenga la necesidad de cargar nuevos valores en el gráfico de línea temporal.

A la hora de pintar los datos reales, lo que se hacía era leer un número de registros del endpoint, ese número N coincidirá con el que se haya asignado al modelo del widget y lo que se obtendrá en realidad, son los últimos N valores. La cuestión es que esos valores se irán introduciendo en el listado de valores a mostrar en el orden inverso al que se quieren visualizar, es decir, aparecerán en la gráfica el último registro, el penúltimo registro, el antepenúltimo registro, etc. y no sólo eso, sino que ocurrirá exactamente lo mismo con los timestamp (una vez ya transformados a formato hora, lo cual se hará a través de funciones propias de Java, generando un objeto Calendar del que se obtendrá la hora real del tomado de la muestra) que se utilizarán como datos del eje X. En ambos casos, lo datos saldrán en el orden de registro más reciente al registro menos reciente de la consulta realizada y por lo tanto no cumpliría con los parámetros de una línea temporal. Por ello, lo que se hizo fue ir introduciendo los valores dentro del listado del array en el orden inverso para que así se visualicen en el correcto. Para conseguir este efecto, únicamente se debería acceder a:

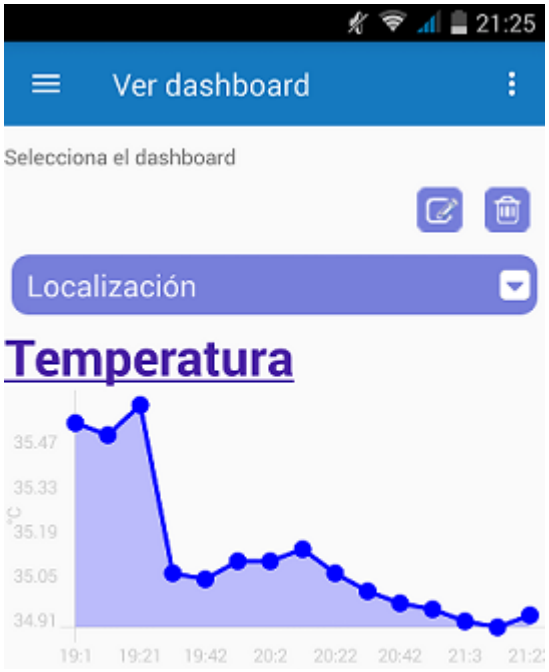

Figure 45: Fragmento LineChartWidget segunda versión

```
listadoValores[tamaño de la lista – índice del bucle]
```

En lugar de:

```
listadoValores[índice del bucle]
```

Con estas modificaciones y añadiendo la etiqueta de 'Tiempo' como unidad de medida, ya se habría solucionado el problema del eje X, pero faltaba el literal y los valores del eje Y. Como no se contaba con demasiado espacio se decidió aumentar la altura del componente, pero aun así fue necesario limitar el número de entradas del eje. Para ello, se calculó una serie de tramos para el mismo, utilizando para conseguirlo los valores menor y mayor de la línea temporal generada, ya que el eje debe mantener sus asignaciones entre ambos valores. Una vez que ya se sabía cuál era el rango en el que los valores recogidos oscilaban, se calculaban 5 intervalos iguales y se asignaban directamente al eje. Por último, se tenía que añadir la unidad de medida del

widget o como también se ha nombrado anteriormente, el literal del eje, que en este caso formaría parte del modelo de datos del LineChartWidget.

El modelo de datos del widget cuenta con las siguientes propiedades:

- Id → campo textual aleatorio generado como UID. Será un identificador del widget que se usará para localizarlo en un listado de widgets. Su valor en la plataforma será único
- Nombre → nombre del widget
- Subtítulo → subtítulo o descripción textual informativa del widget
- Número de resultados → número de registros que quieren visualizarse
- Dato que mostrar → dato del endpoint que se consultará desde el widget
- Literal del eje Y → literal que representa las unidades de medida del gráfico en el eje Y

3.3.4. Fragmento MapsWidget o widget de mapas

Este tipo de widget únicamente mostrará un mapa de google basándose en la latitud y longitud de los datos que reciba y en caso de no contar con las coordenadas necesarias situará la posición en el punto (0, 0). Es decir, este widget siempre mostrará datos independientemente de que cuente o no con ellos

Figure 46: Fragmento MapsWidget

Para generar el widget de los mapas de Google se siguieron las instrucciones proporcionadas por la guía oficial de Google que puede encontrarse de manera online y gratuita. En esta guía se comentaba que lo primero era descargar en el programa AndroidStudio los paquetes de GooglePlay Services, incluir después la librería en el Gradle y sincronizar el proyecto.

El siguiente paso era crear un proyecto con un Maps Activity, lo cual no aplicaba exactamente a las necesidades del proyecto, pero que fue útil al crearlo independientemente para poder estudiar el código que genera automáticamente el propio programa.

Después de estudiar el código se observó que se debía modificar el archivo Manifest.xml para añadir un nuevo permiso de localización y una nueva clave de acceso a las APIs de Google Maps, para así poder realizar todas las consultas que se necesitasen. Esa nueva clave de acceso podía generarse manualmente y añadirla a una cuenta de 'Google Play Console', que es desde dónde se pueden gestionar todas y cada una de las APIs públicas de Google, o por el contrario se puede utilizar para ello el enlace que proporciona autogenerando un Activity de Google Maps. Ese enlace tiene en cuenta la aplicación con el paquete de desarrollo y la función de cifrado SHA-1 con clave privada que utilizará para generar la clave de las credenciales de la aplicación por su cuenta. Finalmente esa clave deberá ser habilitada para conseguir activar los permisos de accesos desde la propia aplicación y se contribuirá en el fichero Manifest.xml para que aplique a la aplicación por completo. Por supuesto, el primer punto se realizará desde la consola de Google.

Además de añadir todas las áreas típicas que se han ido incluyendo en el resto de layouts de los widgets, como pueden ser los paneles de los mensajes de aviso, las etiquetas de títulos, subtítulos y otros campos del widget, de la sección de iconos de edición y borrado y del formulario de edición, era necesario incluir un nuevo elemento similar al que se podía encontrar en los widgets de línea temporal, ya que al fin y al cabo es un módulo de una librería externa. Este elemento propio de la librería puede ser observado a continuación:

```
<fragment xmlns:android=http://schemas.android.com/apk/res/android
xmlns:map=http://schemas.android.com/apk/res-auto
xmlns:tools=http://schemas.android.com/tools
android:id="@+id/map"
android:name="com.google.android.gms.maps.SupportMapFragment"
android:layout_width="match_parent"
android:layout_height="350dp"/>
```

En ese elemento del layout se cargará el mapa una vez que la función OnMapReady() finalice la llamada a la API de Google. Esta función forma parte de la interfaz de Google Maps y se implementará en la clase, para que una vez se cuente con el mapa, sea posible situar un marcador basándose en los datos de latitud y longitud recibidos en el Parcelable. Además de ese marcador se deberá hacer uso de las funciones propias de Google Maps y mover la cámara hasta esa posición, o en otras palabras se desplazará el área visible del mapa a las coordenadas del marcador. No sólo se deberá desplazar la posición del mapa, sino que será necesario animar ese desplazamiento para aplicar un zoom, que permita ver el mapa mejor. En este caso se ha decidido elegir un zoom de 14, para que sea más sencillo el situarse. Si no se incluye el zoom, el marcador no se sabrá exactamente adónde apunta porque quedará a demasiada altura.

Por último, cabe destacar que en el desarrollo se han encontrado algunos problemas de autenticación con la clave cuando se realizaban las baterías de pruebas con otros dispositivos Android. El error que arrojaba era que no se encontraban autorizados a utilizar la 'Google Maps Android API v2' y que era probable que la clave configurada no estuviera habilitada, lo cual no podía ser, ya que en otros dispositivos funcionaba a la perfección.

Después de mucho investigar, se descubrió que la solución para esto consistía en disminuir las restricciones y limitaciones del acceso a la API a través de las credenciales generadas.

El modelo de datos del widget cuenta con las siguientes propiedades:

- Id → campo textual aleatorio generado como UID. Será un identificador del widget que se usará para localizarlo en un listado de widgets. Su valor en la plataforma será único
- Nombre → nombre del widget
- Subtítulo → subtítulo o descripción textual informativa del widget
- Latitud → coordenada que se mide desde la línea del ecuador hacia arriba o hacia abajo
- Longitud → coordenada que mide la posición este y oeste

3.3.5. Fragmento ImageWidget o widget de imágenes

Es un widget que permite cargar imágenes para acompañar a los dashboards y que las ajustará según la medida del terminal móvil.

Al igual que el resto de widgets los datos que se mostrarán se cargarán en el widget utilizando aquellos que se encuentren almacenados en el dispositivo móvil y transferidos entre fragmentos por el Parcelable, pero para obtener la URI de acceso a la imagen se necesitaban realizar una serie de pasos que vimos en el fragmento AddImageWidget, ya que la explicación del Bitmap que se encuentra en ese apartado, es igual en este caso.

Figure 47: Fragmento ImageWidget

Finalmente y realizando distintas pruebas se descubrió que la aplicación no era capaz de cargar algunas de las imágenes que el usuario tenía en la galería de imágenes del dispositivo. Revisando las trazas que se obtenían en AndroidStudio, se vió que el error que devolvía era que que el Bitmap era demasiado grande para ser cargado en la aplicación.

Para solucionarlo se investigaron las opciones que proporcionaba la documentación oficial de Android, ya que son completamente conocedores de esta situación y la respuesta que daban a este problema pasaba por utilizar alguna librería externa, como por ejemplo la librería Picasso. De hecho, ellos mismos proporcionaban el nombre de algunas de estas librerías, incluida la que se ha escogido para el desarrollo.

Además de los pasos habituales de añadir la librería al Gradle y de la sincronización de esta en el proyecto, se necesitaba calcular las dimensiones y proporciones de ancho y alto del terminal móvil, para con ellas poder adaptar el tamaño del Bitmap a cargar.

La carga de la imagen únicamente necesitará la siguiente llamada a la librería:

```
Picasso.with(this.getContext())
                    .load(uri)
                    .resize(size, size)
                    .centerInside()
                    .into(lblImage);
```

La cual realiza las siguientes acciones. Primero carga la imagen a través de la dirección URI que se tenía almacenada en los datos del widget, después se modificaba su tamaño para que se ajuste a las dimensiones que se han calculado justo antes, después se centraba la imagen en el dispositivo y era incluida en el elemento ImageView identificado con la etiqueta lblImage en el código de la aplicación.

Dentro del layout de '*Añadir widget'* de tipo imagen, además del botón la carga de la imagen se añadió un botón de borrado para que elimine la URI que posteriormente se almacenará en el dispositivo y oculte la anterior imagen cargada.

Otro dato de interés sobre este widget, es que si no se incluye una imagen puede utilizarse como un widget textual, ya que es el único widget que no añade datos extra en el layout si no tiene contenidos reales.

Al realizar las pruebas de funcionamiento de la aplicación en distintos terminales se observó que para el widget de las imágenes no se gestionaba igual y dependía de la versión de Android. Este cambio se produjo a partir de la versión v19 o Kitkat y aunque el acceso a la galería mantenía el mismo

código, no era tanto así, los accesos posteriores a las mismas. Cuando se cargaba una imagen independientemente de la versión de Android y esta era pintada igual que se realizaba en la creación del widget era posible visualizarla sin ningún problema, pero una vez se cerraba la aplicación o se apagaba el dispositivo móvil y se volvía a encender y a abrirla, la imagen desaparecía hasta que se volvía a cargar una nueva.

Aparentemente el problema radicaba en la asignación de permisos, porque a partir de la versión v19, el permiso de acceso a cualquier documento, en este caso imágenes, sólo duraba tanto como dura la sesión de la aplicación por lo que al cerrar la sesión ese permiso era revocado y la aplicación no era capaz de obtener el fichero binario de la imagen. De hecho, el error que arrojaba era que la aplicación y el terminal no eran capaces de encontrar la imagen cuando en ambos accesos, el inicial y el de después de cerrar la aplicación se podía observar que la ruta era exactamente la misma.

La solución por lo tanto era la de la prolongación de la duración del permiso y la primera opción fue la de incluir en el archivo *Manifest.xml* un nuevo permiso de acceso a la galería e intentar extenderlo a las imágenes de manera general, pero no supuso ninguna mejora al problema. Finalmente y después de una investigación al respecto, se observó que la solución idónea consistía en proporcionar ese mismo permiso individualmente a cada imagen, pero únicamente para versiones iguales o superiores al Android Kitkat, para versiones anteriores se podría mantener el desarrollo previo, ya que es el adecuado para gestionar las imágenes en esos dispositivos. Para desarrollar esta condición de versiones se incluyó un bucle condicional de la versión de Android que el código es capaz de leer directamente del terminal, y se añadieron una serie de flags de permisos específicos de lectura a la petición de acceso y carga de imágenes desde la galería. De esta manera al realizar la lectura de la misma se conseguía incorporar el permiso adecuado para poder mantener la imagen visible y accesible en la aplicación en sucesivas aperturas, cierres de sesiones y similares.

Cabe destacar que según se leído en algunas comunidades de desarrolladores el máximo número de documentos a los que se les puede proporcionar este permiso prolongado es 128, por lo que en ningún momento se han superado las limitaciones y restricciones que proporciona la programación en Android. Aun así, se tendrá que tener este dato en cuenta por si en el futuro fuera necesario aumentarlo o modificar algún parámetro extra.

El modelo de datos del widget cuenta con las siguientes propiedades:

- Id → campo textual aleatorio generado como UID. Será un identificador del widget que se usará para localizarlo en un listado de widgets. Su valor en la plataforma será único
- Nombre → nombre del widget
- Subtítulo → subtítulo o descripción textual informativa del widget
- URI de la imagen → ruta donde se encuentra la imagen y que se necesita para acceder a ella

3.4. Comunicación REST API

Este paso en el desarrollo consistía en dejar de utilizar datos mock (simulados) en los dashboards y widgets. Para ello eran necesarias varias tareas. Una de ellas consistía en conectarse a un endpoint (será el punto de acceso) que proporcionara datos reales de un proyecto IoT. Con esta premisa se tuvieron que añadir los permisos para que la aplicación sea capaz de acceder a datos externos que se encuentren en internet. Para ello fue necesario añadir la siguiente etiqueta en el fichero AndroidManifest.xml que se encarga de gestionar los permisos que se necesitan desde la aplicación:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Una vez declarada la necesidad de acceso a internet era necesario modificar los modelos de aquellos widgets que accedieran a datos de un endpoint externo, que en este caso eran el SingleDataWidget y el LineChartWidget para que el modelo de datos recogiera en lugar del valor real del dato, el valor del tipo de dato que se quería consultar. Es decir, que en lugar de recoger el valor '41.0', se almacenaba el valor 'temperatura'. Esto se realizó para que cuando se cree el fragmento que contenga al widget, este realice una consulta al endpoint y obtenga el último o el rango de últimos valores y lo muestre. De esta manera se contará con el valor más actualizado posible.

Esto supondrá que los valores que podrá tomar el atributo 'endpoint' deberán coincidir con los que ofrezca el servidor.

Teniendo ya los modelos actualizados e incluidos los atributos nuevos en las instancias simuladas, se pasó a crear una clase que representase las peticiones al servidor que devolvería los datos, que en este caso se trataba de un API Rest. Esa clase ha sido denominada como GetIoTValues y cuenta con tres métodos que se encargan de acceder a los datos:

- onPreExecute() → es un método que se ejecuta antes de realizar la llamada al endpoint. Puede usarse para preconfigurar datos o cambiar el estado de la vista previo a la recepción de los datos. En este caso se ha utilizado para que se muestre un control Progress Bar con estilo de rueda para que dé vueltas hasta que se cargue el dato. Esto se ha hecho así porque si no se incorpora ese elemento se queda un hueco en blanco y da la impresión de que no está ocurriendo nada, mientras que si se pone este elemento se deja claro que se está ejecutando algún tipo de acción y que aún no está terminada. En realidad el elemento ya existe en el layout de la vista, pero es mantenido oculto a través del valor del atributo 'visibility:gone' hasta que se necesita demostrar que el elemento está consultando datos

- doInBackground() → como el caso del LineChartWidget es un listado de valores lo que se hará tanto para este widget como para el del tipo SingleDataWidget es crear un array con los valores recuperados del servidor, aunque en el caso del SingleDataWidget será siempre un array de un único elemento. La llamada se realizará desde una clase independiente que asigna el tipo de solicitud o método de acceso, que en este caso será 'GET', iniciará una conexión http utilizando como parámetro la url del endpoint de destino y usará un buffer de lectura de los datos para recabarlos todos en una única cadena. La respuesta coincidirá con la proporcionada por el endpoint y la parametrización de las consultas dependerá del propio servidor, como por ejemplo crear una consulta para conseguir un número determinado de registros o en un determinado rango de tiempo (basándose en los timestamps)

- onPostExecute() → son acciones que se ejecutan una vez se han recuperado los datos, es decir, al acabar la consulta. En este caso se ha utilizado esta función para modificar la vista y volver a ocultar el Progress Bar con el valor 'visibility:gone' y a su vez se encargará de modificar el contenido del valor del control en la vista correspondiente. En otras palabras se asignará un nuevo valor a la vista que se corresponderá con el valor obtenido a través de la consulta. Para el caso del LineChartWidget, habrá que generar a través de esos valores la estructura que la librería necesita para crear y mostrar el gráfico

Otro de los detalles interesantes de esta clase es que hereda de la clase AsyncTask, lo cual marca que la acción se realizará asíncronamente. De hecho, al tratarse de una llamada al servidor del endpoint es imposible que no fuera una llamada asíncrona, ya que es necesario esperar a la respuesta del servidor y esta no tiene por qué ser inmediata, de ahí que se cuente con los tres métodos que se han visto en el punto anterior.

A partir de este punto los datos recibidos y mostrados ya eran reales.

3.5. Base de datos

Esta parte trata sobre la escritura de los datos de la aplicación, es decir, de la creación de dashboards y widgets y la persistencia de estos mismos datos en el terminal móvil. Como se pudo ver en la sección del estado del arte, la persistencia de datos puede conseguirse desde distintas variantes y desarrollos bastante dispares entre sí. En esta parte se enumerarán aquellos que se han ido probado a la hora de almacenar los datos.

La primera prueba que se realizó fue con el motor de base de datos SQLLite. Para ello se creó un layout específico con un formulario que recogiera los datos y un botón que activara la acción de almacenaje. Con SQLLite fue necesario crear una sentencia SQL de creación de la tabla correspondiente, que para este caso era la de dashboards, y únicamente se crearía si no existía previamente.

```
"CREATE TABLE Dashboards(id INTEGER, nombre TEXT, principal INTEGER)";
```

Para el campo 'principal' o 'primary' se utilizaba un INTEGER, es decir, un valor numérico que consistirá es los valores 0 o 1, ya que no permite almacenar valores booleanos como true o false. Esto significa que previo al guardado del dashboard se debería realizar una transformación del atributo que se recogiera de la instancia.

A continuación, se tenía que acceder en primera instancia a la base de datos que se había creado y justo después de eso, transformar el objeto dashboard para que se ajuste a los campos que aparecen en la definición de la tabla y ejecutar una nueva instrucción SQL usando para ello la función de Android 'db.execSQL' de la manera que se puede observar en la siguiente sentencia:

```
db.execSQL("INSERT INTO Dashboards(id, nombre, principal) " +
            "VALUES (" + id + ", '" + nombre + ", '" + principal +")");
```

Una vez en este punto, se vio en seguida que el código resultante era algo confuso y difícil de interpretar cuando únicamente se contaba con dashboards que ni siquiera tenían el listado de widgets, así que en vista de que en el futuro se debía añadir esa otra parte y que ya daba la impresión de que era complicado de mantener, se decidió que era mejor optar por otro camino, por lo que se investigaron alternativas para ello y se encontró el uso de ORM. Como se comentó en el apartado del estado del arte, los ORM se encargan de transformar un sistema de datos orientado a objetos como es el utilizado en los desarrollos Android en Java a un sistema relacional como lo es SQLLite. La librería que se seleccionó para ello fue Realm para Android, cuya sintaxis era mucho más simple y fácil de seguir e interpretar, haciendo el trabajo mucho más sencillo y cómodo, pero en etapas tempranas del desarrollo se descubrió que no era posible seguir este camino, al menos no con esta librería, ya que no soportaba alguna de las funcionalidades que se necesitaba para el proyecto, como por ejemplo la herencia y el polimorfismo, y no parecía que fueran a trabajar en esta línea en un corto periodo de tiempo, así que fue necesario investigar otras alternativas que se ajustasen a las necesidades actuales.

Entre esas alternativas que se barajearon, se pensó en generar un Json que contuviera los datos y persistirlos en este formato. Esto significaba que se debía no sólo hacer una transformación de los objetos a formato Json, sino que era necesario ver y estudiar si se almacenaban los datos en un fichero interno o externo, si se elegía la opción de conexión de red o si se utilizaba con las preferencias compartidas. La primera opción descartada fue la de la conexión de red, ya que no sólo implicaba un desarrollo extra en la parte del servidor, sino que las ventajas de compartición y acceso externo que ofrece esta alternativa no encajaban a priori con lo que se necesitaba en este proyecto, por lo que se continuó estudiando el resto.

Finalmente se decidió implementar una solución que partiera de los datos con formato Json. Para poder hacer esto, se utilizó la librería Gson que es una librería desarrollada por el equipo de Google, que se encarga de añadir funcionalidad para transformar objetos Java a Json y viceversa. Los pasos de este desarrollo consistían en crear una nueva clase que representara el modelo de datos de la

aplicación por completo, en otras palabras, que contuviera el listado de dashboards. Esa clase Java se nombró como AppDataModel. Al contar con esta clase y con la librería Gson, ya se era capaz de tener un objeto de tipo AppDataModel que tuviera el listado de los dashboards y que a su vez cada uno de estos dashboard tuviera su propio listado de widgets, recogiendo así todos los datos que componen la aplicación.

La necesidad de tener esa clase era principalmente para poder transformar todo el sistema en un único Json que lo englobara y así utilizar Gson para la transformación de objeto a Json y viceversa de manera trivial. Una vez hecha la transformación del objeto a formato Json, se podía sin más preámbulo pasarlo a String para almacenarlo en las preferencias compartidas, de tal manera que fuera menos costoso acceder a los datos que si de un fichero se tratara. Con estas pequeñas modificaciones ya se mantenía en la aplicación una base de datos no relacional que además se asemejaba a la existente en la aplicación de escritorio.

En este punto ya se podía definir la lógica del almacenaje de los datos del dashboard. El primer paso era comprobar si el usuario ya contaba o no con algún dashboard. Si no lo tenía, se forzaría ese dashboard como panel principal y a continuación se guardaría. Si sí que tenía algún dashboard, comprobaría si el nombre seleccionado para el nuevo ya se estaba utilizando en alguno de los existentes, si el nombre ya estaba en uso no permitiría guardar el nuevo. Si no se estaba usando, faltaría una última comprobación que consistía en asegurarse de si se había marcado el dashboard como principal o no. Si no se había marcado como el principal se almacenaba, pero si se había marcado como el principal se avisaba desde el terminal de que ya existía uno como principal y que si quería que perdiera esa preferencia o no. Si el usuario escogía el cambiar el dashboard principal, se guardaba el nuevo tal cual se habían introducido sus datos y se modificaba aquel que estaba marcado anteriormente como primario. Sin embargo, si el usuario marcaba que no quería hacerlo primario, entonces se cambiaba el registro a ser almacenado para que ya no marcara la opción de principal y por lo tanto pudiera ser almacenado sin problemas en el dispositivo móvil.

Ese objeto transformado en Json con la librería Gson era correctamente almacenado en las preferencias compartidas utilizando el siguiente código:

```
SharedPreferences.Editor editor = settings.edit();

jsonArray = gson.toJson(appDataModel, AppDataModel.class);

editor.putString("dataBaseModel", jsonArray);
editor.apply();
```

Mientras que la lectura de las preferencias compartidas es tan sencillo como leer un String con el nombre concreto con el que se guardó el Json con anterioridad:

```
//Shared Preferences
SharedPreferences settings =
PreferenceManager.getDefaultSharedPreferences(this);
String jsonArray = settings.getString("dataBaseModel", "");
```

El problema llegó cuando se llegó a la fase del almacenamiento de widgets, porque como se ha comentado en los apartados anteriores sobre el desarrollo, la clase que se utilizaba para la generación del listado era la interfaz '*Widget'*. Esto significaba que al transformar el listado a formato Json todos los objetos que representan a los widgets saldrían como instancias de la clase 'Widget' la cual al ser una interfaz no puede instanciarse, además era imposible saber de qué tipo era cada widget para poder instanciarlos de manera individual.

Debido a esto, fue necesario crear un serializador y deserializador propio basándonse en la librería Gson para que fuera compatible con esta. Para el serializador se recorrería el objeto AppDataModel

empezando por el listado de dashboard y continuando por el listado de widgets, una vez llegado a este punto, se obtenía el objeto widget y su clase y se transformaba en un elemento Json formado por dos partes:

- Nombre de la clase del objeto widget – por ejemplo
    - 'uc3m.pfc.thingdroid.model.implementation.ImageWidget'
- Resto de la instancia del widget o en otras palabras el resultado de la serialización normal del objeto

Ambos datos formarían un Json similar al que se puede ver en este ejemplo:

```
{
	"CLASSNAME":"uc3m.pfc.thingdroid.model.implementation.LineChartWidget",
	"INSTANCE":{
		"axis":"ºC",
		"endPointName":"temperature",
		"id":"f5d6b404-c511-4390-8eec-ac3c9b95a0b9",
		"title":"Temperatura",
		"subtitle":"",
		"numberOfValues":15
	}
}
```

Este nuevo objeto formado por CLASSNAME e INSTANCE se añadirá al JsonArray de widgets y este array se incluiría en el elemento JsonElement dashboard conjuntamente con las características del objeto dashboard, es decir, id, nombre y el booleano de dashboard principal. Este mismo procedimiento se aplicará para todos los dashboards y se generará un JsonArray de ellos. Por último se genera el JsonElement que representará al objeto AppDataModel que será el resultado de la serialización.

Por el contrario, el proceso de deserialización se encargará de generar un objeto AppDataModel a través de un JsonElement de origen, es decir, que realizará la operación inversa generando objetos de las clases basándose en la estructura del Json y a la hora de llegar a la instanciación de los widgets, se utilizará la propiedad CLASSNAME que se vio anteriormente para conseguir la instanciación normal del widget utilizando las clases genéricas de deserialización:

```
String className = widgetAux.get(CLASSNAME).getAsString();
Class<?> classObject;
try {
    classObject = Class.forName(className);
} catch (ClassNotFoundException e) {
    e.printStackTrace();
    throw new JsonParseException(e.getMessage());
}

context.deserialize(widgetAux.get(INSTANCE), classObject);
```

Estas funciones serían compatibles con todos los tipos de widgets y no necesitarían modificación alguna cuando se añadieran los nuevos tipos, ya que la instanciación de esos objetos los realiza de forma genérica a través de la propia deserialización del lenguaje de programación Java.

A partir de este punto, se debía utilizar la librería Gson a través de las clases complementarias de

GsonBuilder a las que se agregaría el adaptador de serialialización/deserialización que se han visto en este apartado. Una vez se instanciaba el objeto Gson con estas características era posible usarlo de la manera habitual con las funciones ‘toJson’ y ‘fromJson’ que simplificaban en gran manera el desarrollo de la persistencia de datos, ya que se escrbirían los datos en la aplicación en formato Json, pero luego se gestionarían desde la misma con los objetos de las clases propias que se han desarrollado principalmente por la simplicidad del manejo de los datos y la flexibilidad que aporta al proyecto.

Esta parte de la aplicación habría que aplicarla en todas y cada una de los Activity y fragmentos que se encargasen tanto de mostrar o almacenar datos propios de la aplicación, para que siempre mantenga el estado de la misma y pueda observarse la continuidad.

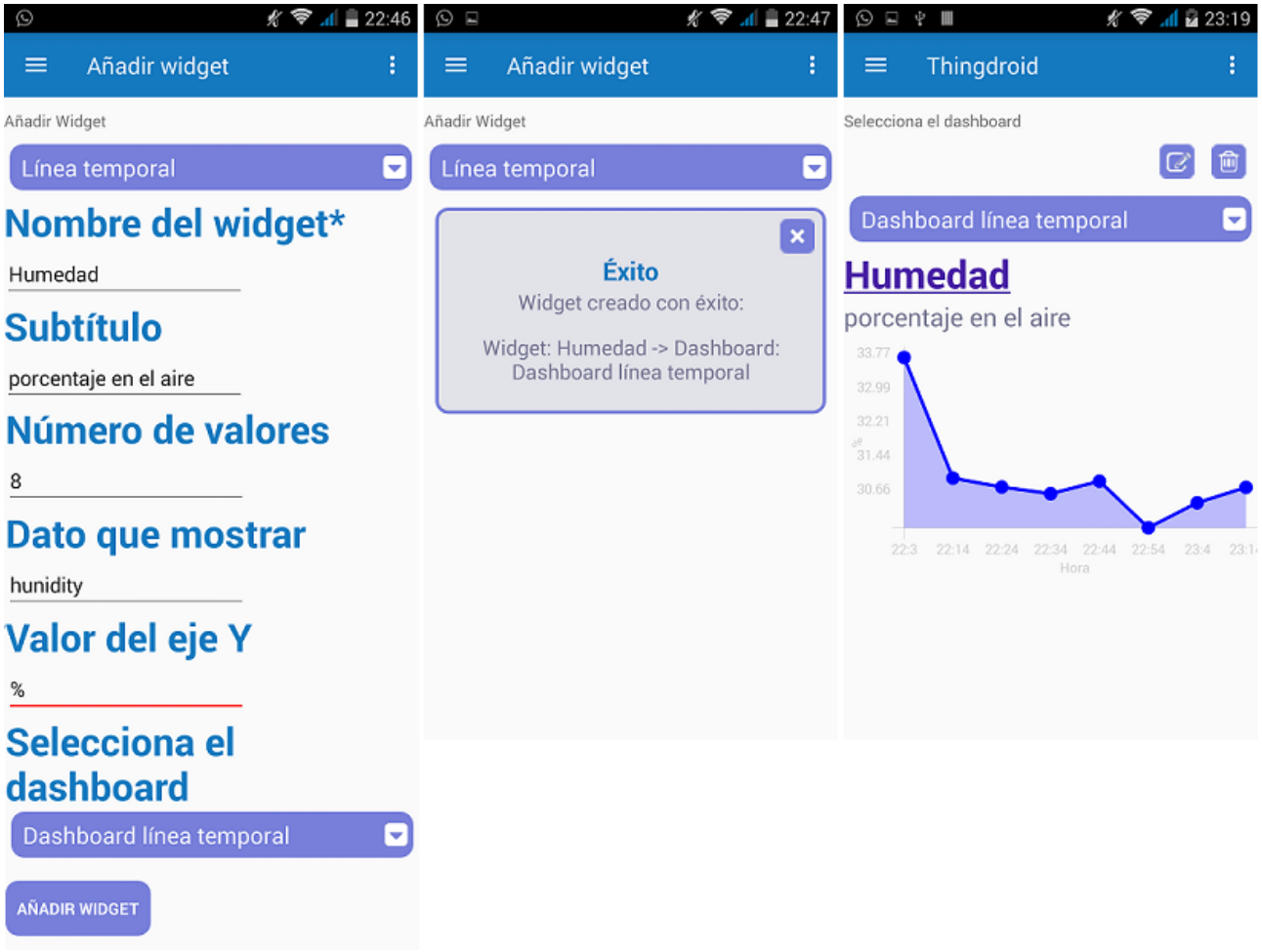

Figure 48: Almacenaje de un widget y su posterior lectura

3.5.1. Modo eliminación

Una vez terminado todo el desarrollo de la persistencia de datos, era esencial dar la posibilidad de eliminar esos datos. La aplicación permite de por sí, eliminar todos y cada uno de los datos almacenados en el terminal a través de los ajustes de Android al igual que cualquier aplicación al uso, y para esto no es necesario ningún tipo de desarrollo extra porque es algo que gestiona directamente Android por su cuenta.

Pero este tipo de eliminación de datos acaba con todos y cada uno de ellos y lo que se quería hacer era una eliminación selectiva.

Primeramente segestionaba el borrado de dashboards completos. Para ello, se incluían nuevos iconos con forma de papelera y se le asignaban eventos de click sencillo para poder activar las acciones que se querían que se llevaran a cabo. Nada más hacer click en el icono aparecerá un mensaje de aviso para prevenir errores de clicks accidentales o involuntarios. Este mensaje obtendría sus textos

directamente del archivo strings.xml. Si en ese panel de aviso se confirma el interés del usuario por la eliminación del dashboard, será tan sencillo como obtener el objeto dashboard, lo cual puede hacerse con el nombre del mismo, ya que es único y está accesible desde la opción fijada del spinner de la pantalla '*Ver dashboard'*. Luego con el objeto dashboard encontrado y habiendo anteriormente generado el objeto de AppDataModel que contiene el listado de dashboards (por supuesto, esta información estará sacada de las preferencias compartidas y se encontrará actualizado con el último estado de las mismas), se podrán utilizar los métodos propios de su clase con las funciones nativas de Android de gestión de listas. Además de eliminarlo de esa instancia AppDataModel comprobará si el dashboard eliminado era el principal, si no lo era, el desarrollo quedaba tal cual, pero en caso contrario se debía seleccionar un nuevo dashboard como principal y en este caso se decidió elegir el primero del listado por defecto, siempre teniendo en cuenta que se tenga algún dashboard adicional en el listado al que hacer principal. El valor principal se asignará con la función setter correspondiente sobre la instancia del objeto.

Por último se deberá actualizar el estado de las preferencias compartidas para mantener la coherencia de la aplicación, así que se escribirá el nuevo Json en la misma utilizando la instancia de AppDataModel que se ha estado modificando y se actualizará la vista de 'Ver dashboard' para que recargue en el spinner el resto de dashboard con los que aún existen.

Esta explicación era únicamente para los dashboard, pero debía ser extendida a los widgets. El proceso sería similar, se contaría con un icono propio en la vista del widget, que al hacerse click sobre él mostraría un mensaje de aviso para asegurarse de que la acción ha sido propiciada correctamente por el usuario. Una vez llegados a la confirmación, se debería conseguir primeramente el nombre del dashboard, lo cual se haría a través de la información de los Parcelables con los cuales se enviaban datos desde un fragmento a otro. Por ello, además de enviar la información propia del objeto widget, se transfería el nombre del dashboard en la nueva variable textual 'dashboardData'. Con este dato ya era posible incluir un nuevo método en la clase AppDataModel que con el nombre del dashboard pudiera acceder a su instancia y ahí eliminar de ese objeto, el widget cuya propiedad 'id' coincida con la del widget que se quiere eliminar. Por supuesto, la función de eliminación de un widget del listado se hará con las funciones nativas de Android, o lo que es lo mismo, de Java. En este caso, la gestión se hará con los métodos de la clase Dashboard.

Para terminar y al igual que en el caso de los dashboard, se actualizarán las preferencias compartidas y se volverá a renderizar la vista 'Ver dashboard', pero en este caso al no pertenecer el icono de borrado a la propia vista de 'Ver dashboard' sino a la del propio widget, éste deberá comunicarse con su fragmento padre y solicitar esa recarga de la vista. Para ello, era necesario encontrar el fragmento padre, lo cual lo se conseguiría a través de la siguiente función Android:

```
ShowDashboard showDashboardFragment =
(ShowDashboard)getFragmentManager().findFragmentByTag("ShowDashboard");
```

El etiquetado que utiliza para la búsqueda del fragmento también fue necesario incluir en el momento de la creación del mismo, lo cual ocurre cuando se selecciona esa sección en el menú principal y se reemplazan los fragmentos con las vistas correspondientes.

Con el fragmento encontrado, se debía llamar a un método que tuviese la clase que se encarga de gestionar esa vista y que la actualizase y solicitara de nuevo el renderizado. Por supuesto, ese método debía mantener los datos de la instancia AppDataModel actualizados por lo que debía leer de nuevo las preferencias compartidas, que contendrían la misma información que antes menos el widget eliminado. Además, si el widget se borraba de un dashboard que no era el principal se debía forzar la selección del spinner a la opción del dashboard, ya que si no se realiza esto, la vista no cambiaría para mostrar el dashboard principal y para poder observar correctamente el borrado, el usuario tendría que volver a desplegar el spinner y escoger el dashboard anterior. Si no se realizase este forzado en el spinner, el usuario quedaría muy confuso, ya que perdería la navegación en la que se encontraba inicialmente.

La actualización de la vista suponía volver a destruir todos y cada uno de los fragmentos de los widgets y volver a cargarlos en el nuevo orden establecido. Es decir, sin el dashboard eliminado los widgets siguientes a éste tienen que adelantar una posición en el listado.

Esta función que se encargaba de la actualización de la vista '*Ver dashboard'* y de todo el tema del borrado de widgets debía hacerse por cada uno de los widgets existentes por lo que fue necesario la actualización de ellos en este sentido que comprende no sólo funcionalidad y código Java, sino también cambios en el layout de cada uno de ellos.

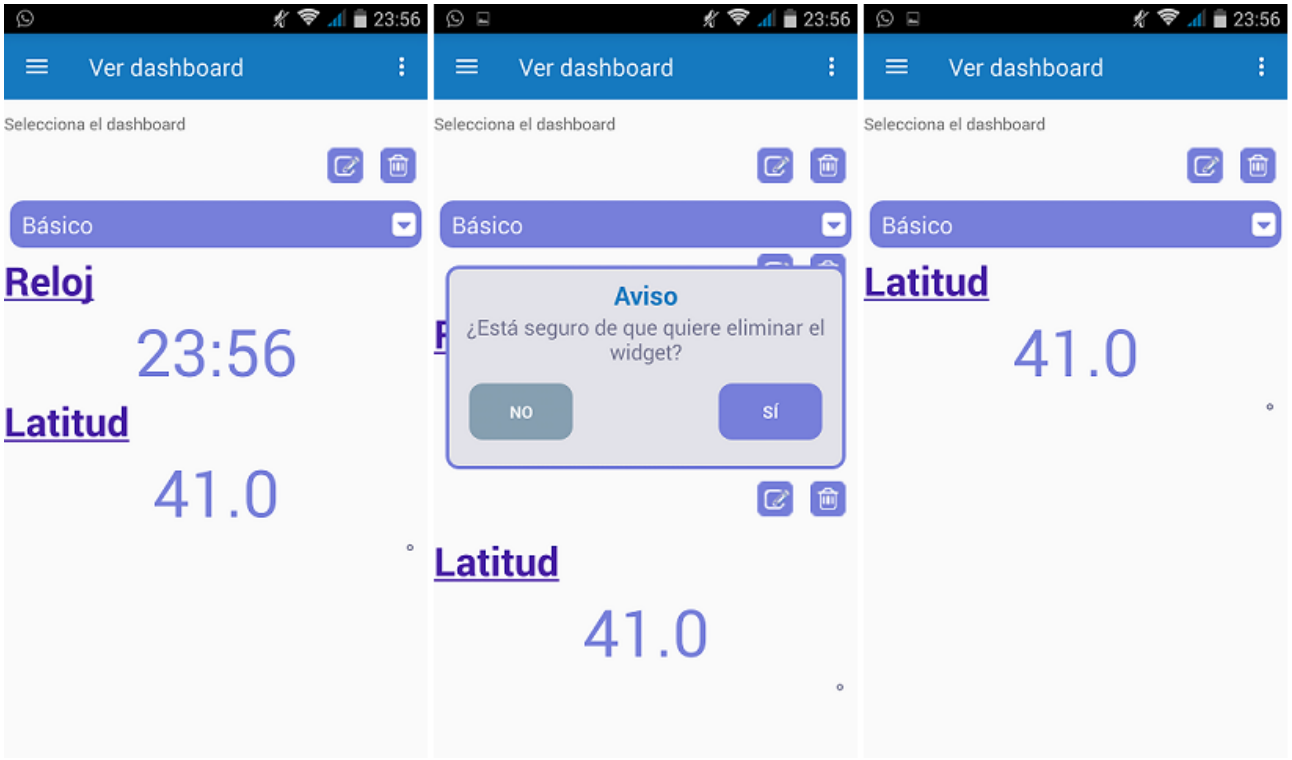

Figure 49: Flujo de proceso de eliminación

Es importante recalcar, que cuando se habla de clicks en realidad son eventos touch en la pantalla del dispositivo, pero además de que la nomenclatura oficial hable siempre de clicks, nombrarlos así en este documento es más intuitivo, ya que pueden compararlo con el lenguaje habitual que se usa a diario en el desarrollo web, que aunque no son exactamente lo mismo puede ser muy útil para situar al lector en el contexto.

3.5.2. Modo edición

Al igual que en el apartado anterior, otra funcionalidad básica que se debía tener en la aplicación era la de la edición de los datos y siguiendo los mismos pasos que en el borrado y eliminación se comenzaría gestionando la botonera con los iconos, en este caso un lápiz, que represente la modificación.

Este icono activará y desactivará el modo edición según sea pulsado, ya que de primeras las opciones de edición estarán ocultas para el usuario. En el caso de los dashboards, la edición únicamente puede hacerse a nivel de la propiedad de principal, si el dashboard ya es el principal el botón para transformarlo no existirá, mientras que si no lo es se mostrará. Para modificar esta opción únicamente se tenían que aplicar las mismas acciones que se hacían cuando se borraba el dashboard principal en el apartado anterior, pero en lugar de forzar a que fuera el primero el principal había que modificar la opción para que cambiara la instancia del dashboard para que fuera el actual. Además de hacer que el dashboard fuese el principal se debía hacer que el anterior principal dejara de serlo, ya que desde el inicio del desarrollo se definió esta propiedad para que únicamente uno de los dashboard fuera el principal porque si todos los dashboard pudieran ser principales al mismo tiempo, eso significaría que

ninguno es realmente principal, o en otras palabras, si todo es principal, entonces nada es principal. Los métodos set de la clase Dashboard, en concreto el setPrimmary() será el que se utilice para este desempeño.

Por otra parte, además de modificar la propiedad de principal del dashboard, también activará el modo edición en los widgets o lo que es lo mismo, mostrará y ocultará para cada una de las vistas de los fragmentos de los widgets la sección de edición/eliminación del widget, que estará formada por los mismos iconos con los que se contaban para los dashboard, es decir, el lápiz y la papelera. En el apartado anterior, la papelera que acompañaba al fragmento de widget estaba visible en todo momento, pero era mucho más limpio e intuitivo que esta sección fuese gestionada en su conjunto, manteniendo la ocultación y la aparición del mismo dependiente del modo edición del dashboard.

Una vez con el modo edición de dashboard activado, es posible a su vez modificar los datos del widget. Para ello y siguiendo el ejemplo anterior, el icono de edición de cada fragmento activaría a su vez el modo edición de widget. Esta acción no supondría otra cosa que mostrar un formulario de edición basado en el de creación del widget correspondiente, pero en lugar de estar vacío, se rellenaría con los valores obtenidos del Parcelable o lo que es lo mismo, con los valores del widget. En ese formulario que imita a los de la sección 'Añadir widget', se pueden encontrar exactamente los mismos campos a excepción del que permitía seleccionar en qué dashboard se debía añadir el widget, ya que se definió que para mover un widget de un dashboard a otro se puede conseguir esto mismo creando un nuevo widget desde cero y borrando el anterior.

Mientras que la edición de los campos 'título' y 'subtítulo' podía realizarse desde los métodos de la interfaz 'Widget' llamándolos desde un método propio de la clase AppDataModel, ya que son comunes a todas las clases de widgets. Para el resto de campos, como por ejemplo 'las unidades de medida' o 'número de valores que mostrar', se debían generar nuevos métodos que se adaptaran a lo que se podría encontrar en cada uno de los tipos de widgets porque no es posible hacerlo desde la interfaz, a no ser de que se decidiera que todos los métodos de todos los tipos de widgets debían aparecer en la interfaz, pero eso no sólo haría perder el sentido de la interfaz, ya que es para información común y esta tendría una cantidad desproporcionada de datos no comunes, sino que supondría que en cada una de las clases que definen los widgets se debería contar con todos los métodos que no se aplicasen a esa clase, vacíos. De tal forma que todos los widgets podrían contener la información del resto, algo que por supuesto no tiene ningún tipo de sentido.

Los dos métodos de edición de 'título' y 'subtítulo' tendrían por lo tanto que encontrar el dashboard concreto al que afectan los campos, y dentro de ese dashboard se recorrerá la lista de widgets hasta hallar el que deba ser modificado, en ese punto se llamará a la función de la interfaz como se ha comentado anteriormente.

Esos nuevos métodos que se debían desarrollar, partían de la misma base que los creados para los campos 'título' y 'subtítulo', es decir, se localizaba el widget dentro de un dashboard concreto, pero como no es posible utilizar los sets propios de las clases particulares de los widgets sobre un objeto de una lista que contendrá la interfaz, se debía almacenar ese elemento encontrado en un objeto del tipo al que se quería acceder. Es decir, que si se querían modificar las 'unidades de medida' se asignarían a un objeto de tipo SingleDataWidget el elemento encontrado en el listado que coincidía con el que se estaba editando, realizando el casting de datos correspondiente, y sobre esa nueva instancia se realizaría la modificación del campo, ya que esta clase sí tendrá el método set necesario para ello. Ahora y con esa instancia independiente se procedería al borrado del widget existente dentro del listado de widgets y se añadiría esta nueva versión del mismo en la misma posición que su predecesor. Por supuesto, para poder hacer esto último, se debía obtener la posición antes de haber eliminado el widget y además hay que tener en cuenta que no es operativo eliminar el widget si se está recorriendo el listado de los mismos porque se producen errores de concurrencia.

Estos métodos deberán crearse para todos y cada uno de los campos editables de cada widget y se llamarán cuando se haga click sobre el botón de 'Guardar cambios' con los datos obtenidos del

formulario.

Después de este cambio de la instancia AppDataModel y como siempre, se actualizarán las preferencias compartidas y se volverá a renderizar la vista completa de '*Ver dashboard'* mostrando ahora la versión actualizada de los datos.

Aunque es lo lógico y probablemente se dé por hecho, es importante recalcar que si después de una modificación vuelve a activarse el modo edición del widget, los datos que saldrán en el formulario correspondiente serán los actualizados.

Cabe destacar que cada icono de lápiz de cada fragmento de widget, únicamente es capaz de activar el modo edición de ese mismo widget y de ningún otro bajo ninguna circunstancia porque cada fragmento se gestiona independientemente.

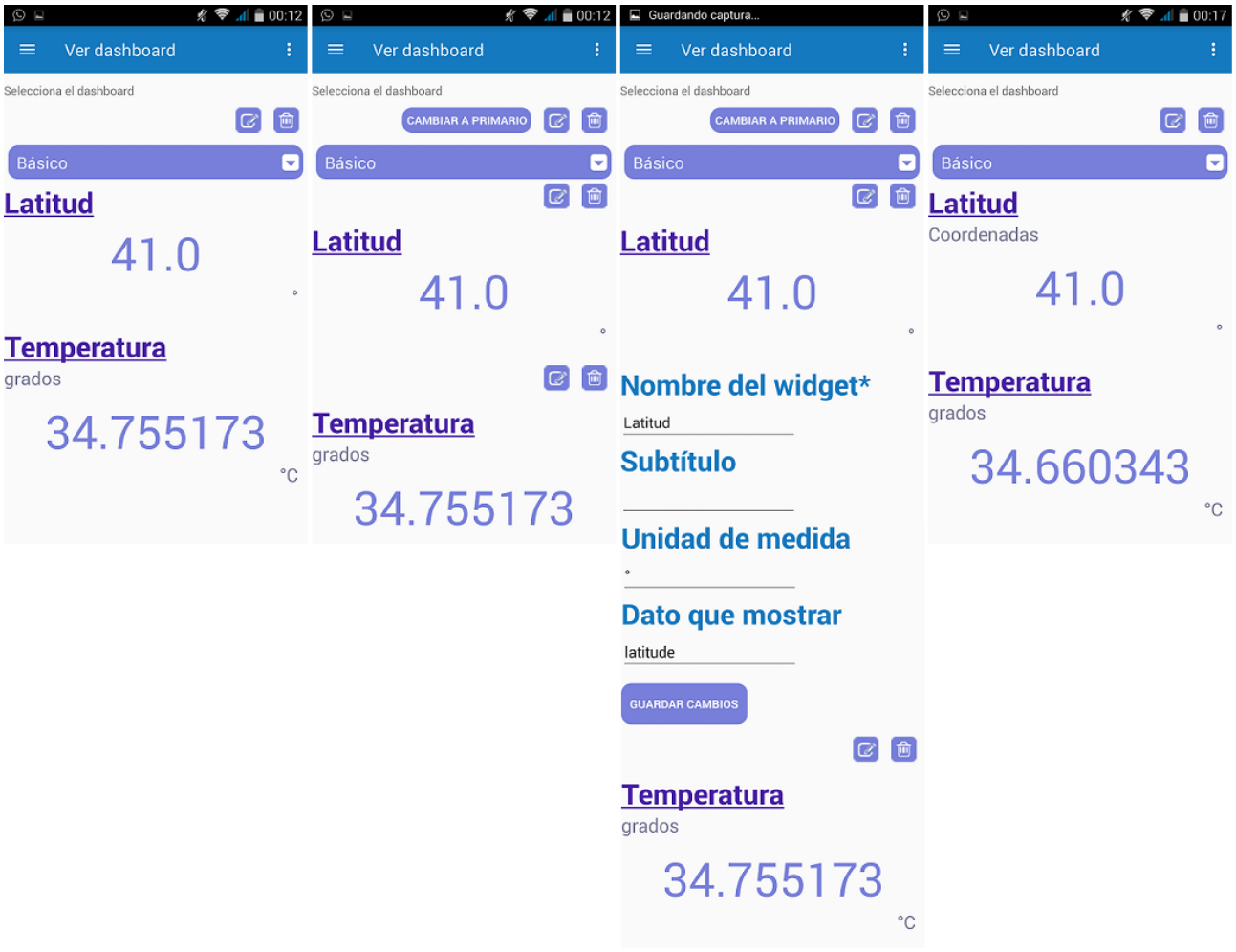

Figure 50: Flujo de proceso de edición

3.6. Desarrollos complementarios

Esta sección se ha incluido para registrar los cambios de código y desarrollo que no encajan en los apartados anteriores.

3.6.1. Cambio de estilos

Al generar todos y cada uno de los formularios fue muy sencillo darse cuenta automáticamente que debían tener un mismo estilo y que la aplicación no tenía un estilo definido y se limitaba a tener un aspecto plano y bastante poco agradable a la vista.

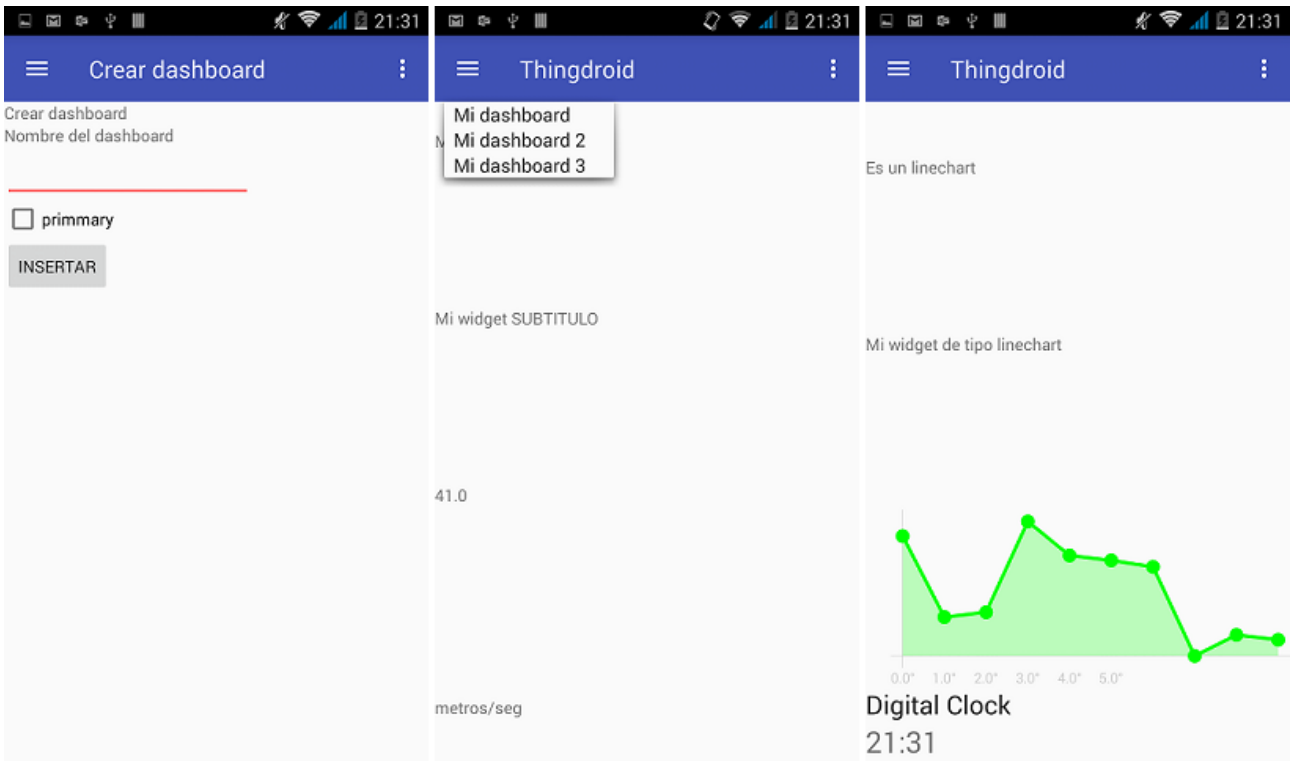

Figure 51: Estilos originales de la aplicación

Debido a todo esto se empezó a trabajar en los archivos de 'Values' para crear estilos que se aplicaran a las etiquetas xml, es decir, a los controles de Android y crear una aplicación homogénea.

Lo primero que se comenzó a modificar fueron los fragmentos que representaban los widgets y se definió un estilo para cada etiqueta de ellos, pero manteniendo el mismo estilo para todas las etiquetas que tenían el mismo significado. En otras palabras, todos los títulos usarían un mismo estilo, al igual que todos los subtítulos compartirían uno propio y las unidades de medidas otro. Este desarrollo es posible observarlo en las siguientes líneas de código que son las que se estaban utilizando en este punto del desarrollo.

Por un lado se temdría el estilo definido en el fichero styles.xml:

```
<style name="widgetTitleStyle">
<item name="android:layout_width">fill_parent</item>
<item name="android:layout_height">wrap_content</item>
<item name="android:textColor">@color/primary_text_color</item>
<item name="android:textSize">15pt</item>
<item name="android:textStyle">bold</item>
</style>
```

Y por otro se borrarían todos los estilos individuales de la etiqueta o control Android y se añadiría una propiedad que enlace ese mismo estilo:

```
<TextView
        android:id="@+id/single_data_title"
        style="@style/widgetTitleStyle" />
```

Como se ha podido observar en el propio archivo de styles.xml y en el ejemplo aportado justo unas líneas más arriba, las variables de los colores estarán definidas en otro sitio, es decir, en otro fichero, el archivo colors.xml cuya sintaxis se puede ver a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<resources>
    <color name="color_primary">#1F303C</color>
    <color name="color_primary_dark">#131e26</color>
    <color name="primary_text_color">#191e5b</color>
    <color name="secondary_text_color">#747591</color>
    <color name="menu_text_color_active">#FFFFFF</color>
    <color name="menu_text_color">#869fb1</color>
</resources>
```

En el caso de estilizar algunos elementos era necesario realizar algunas acciones previas, como por ejemplo con el control 'Spinner'. Para este control se debía desarrollar primero dos layouts uno para el listado de opciones y otro para la opción seleccionada, ya que es prácticamente imposible dar estilos al control por sí sólo, así que se crean y personalizan a través de estos elementos. Con los layouts creados, faltaría asignarlos al Spinner original y esto se haría de manera programática en el archivo Java correspondiente a la vista que contenga el susodicho control. Para ello, se utiliza un adaptador al cual se le asigna el layout de la lista desplegable y el de la opción seleccionable con las siguientes instrucciones:

```
ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(getActivity(),
            R.layout.spinner_item, list);
dataAdapter.setDropDownViewResource(R.layout.spinner_dropdown_item);
spinner.setAdapter( dataAdapter );
```

Después, únicamente es necesario ajustar los estilos en esos layouts y personalizarlos bajo las necesidades del proyecto. En este caso se decidió cambiar la flecha del desplegable añadiendo una nueva imagen y eliminando el fondo del Spinner para que no saliera la que utiliza por defecto. El resultado obtenido fue el siguiente:

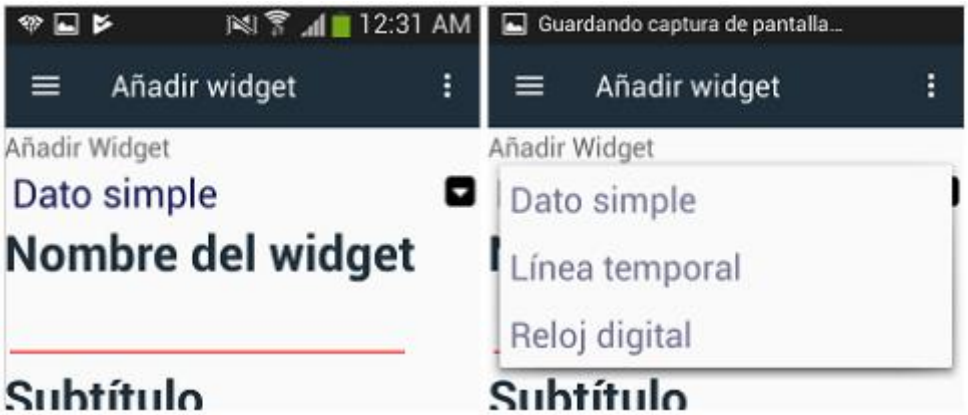

Figure 52: Estilos del Spinner

Este desarrollo que se ha visto en esta sección, no sólo se realizó para estos layouts, sino para todos, no sólo porque deja mucho más limpios y simples los layouts, sino que también simplifica y centraliza su modificación haciendo que el mantenimiento sea mucho más eficiente. Además de esta razón y de la expuesta anteriormente sobre crear un aspecto homogéneo en la aplicación era importante crear unos estilos atractivos para el usuario, ya que hasta este punto del desarrollo el diseño que se estaba mostrando era bastante tosco y para nada lo que se quería para la aplicación final. Este tipo de detalles son los que luego quedan más incrustados en la memoria de los usuarios y aunque pueda parecer algo trivial, el aspecto de la aplicación puede atraer a más usuarios que su propia funcionalidad, porque simplemente por el hecho de tener un aspecto decente y agradable puede conseguir que se le dé una oportunidad que no se le daría de ningún otro modo. En otras palabras, que aunque el código esté

optimizado al máximo, sea totalmente eficiente, con un rendimiento inmejorable y con unas funcionalidades de lo más útiles sin incidencias y libre de cualquier tipo de fallos, si no entra fácilmente por los ojos es probable que no sea utilizada y una aplicación que no se utiliza no tiene sentido.

Como se ha comentado sobre el tema del mantenimiento, es posible ver en la siguiente imagen de ejemplo como cambiando únicamente un par de valores se pueden modificar todos los valores para darle un aspecto más renovado.

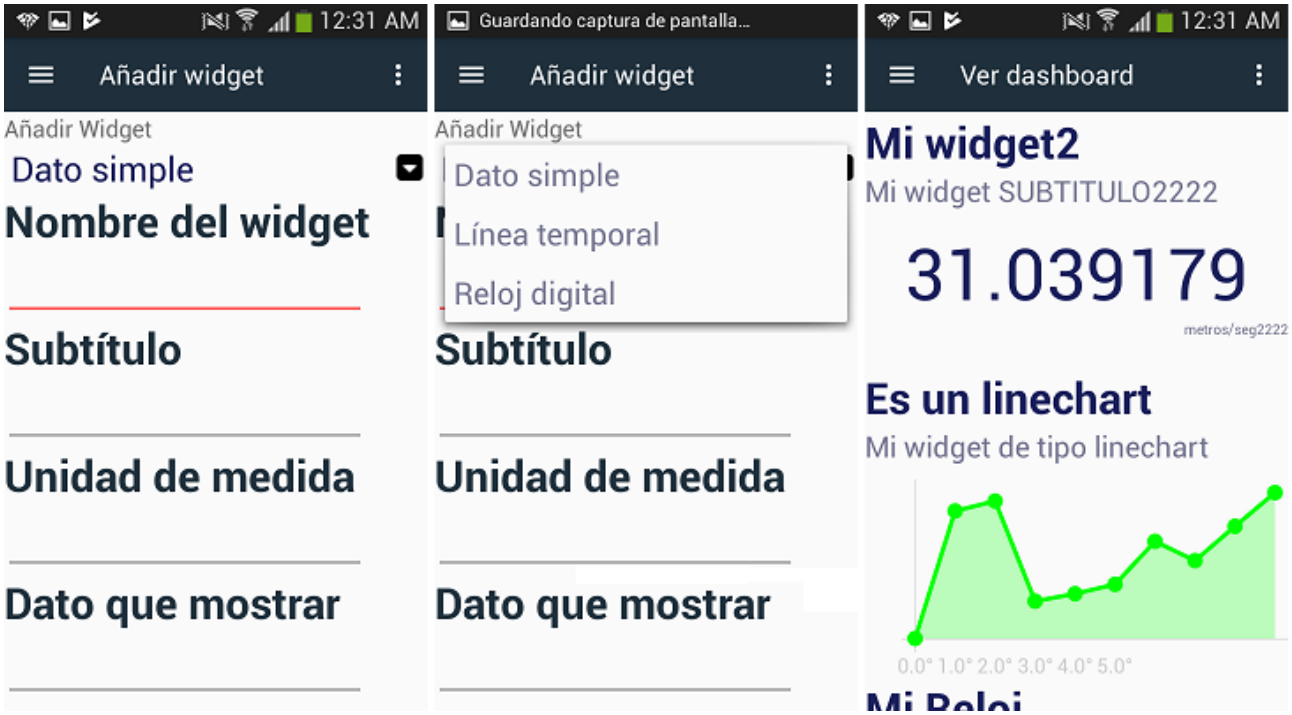

Figure 53: Primer ejemplo de cambios de estilos

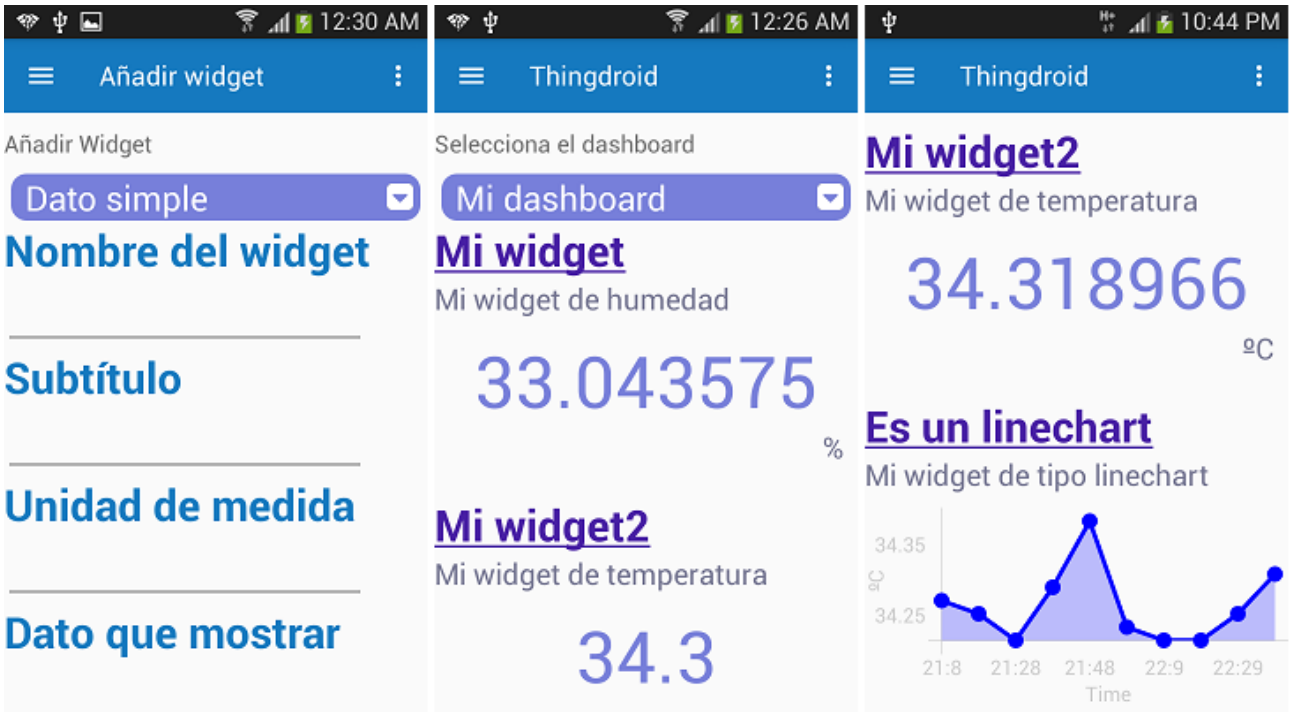

Figure 54: Segundo ejemplo de cambios de estilos

También existe la posibilidad de cambiar el tipo de letra para los campos textuales utilizando el atributo 'android:typeface', pero en este caso se ha decidido no hacerlo principalmente porque usar

fuentes tipográficas que son externas a Android requieren el fichero de la propia fuente, lo cual suele ocupar bastante e incluso en algunos casos se necesita contar con la licencia de las mismas. Teniendo en cuenta que el tamaño de la APK (fichero de aplicación) resultante es una de las mayores preocupaciones de los desarrolladores Android y que siempre se intenta generar la aplicación más reducida posible, las fuentes accesibles para un proyecto habitual serán Droid Sans, Droid Sans Mono y Droid Serif, que en realidad son versiones sans, monospace y serif de la misma tipografía de Android. Aunque cabe destacar que desde la versión IceCream de Android, se ha incluido un nuevo tipo de letra llamada Roboto, cuyo uso está optimizado para pantallas de alta resolución (high definition).

Además de todo esto, es importante que la fuente sea legible y se mantenga estable a lo largo de la aplicación para que cualquier usuario pueda leer los datos fácilmente y sea lo más accesible posible, así que el hecho de utilizar las tipografías por defecto puede ser un factor favorable, ya que tienen en cuenta estos problemas y están diseñadas para intentar paliarlos todo lo humanamente posible, aunque no sólo hay que tener en cuenta la fuente utilizada sino también el contraste de colores y el tamaño de la tipografía.

3.6.2. Cambio de literales

A la hora de poder crear una aplicación que sea multi idioma en Android existe la posibilidad de generar distintos ficheros de strings.xml intercambiables, uno por cada idioma con el que se quiera contar, pero para poder aplicar este cambio es necesario que todos y cada uno de los literales que se encuentren contribuidos en la aplicación se encuentren recogidos en estos ficheros, ya que si no es así, entonces ese texto se quedará tal cual este en el control Android.

Hasta este punto lo que se estaba haciendo era escribir directamente el texto que se quería visualizar.

```
<TextView
       style="@style/sectionTitleStyle"
       android:text="Añadir widget" />
```

Mientras que en el ejemplo siguiente es posible ver cómo se tendría que definir el control ‘TextView’ para poder generar ficheros idiomáticos y cómo tendría que ser el registro del literal en el archivo strings.xml:

```
<TextView
       style="@style/sectionTitleStyle"
       android:text="@string/addWidget" />

<resources>
    <string name="addWidget">Añadir Widget</string>
</resources>
```

Anidados dentro de la etiqueta ‘resources’ se encontrarían todos y cada uno de los literales que se van a necesitar en la aplicación.

3.6.3. Creación de iconos

Para no crear múltiples botones que ocuparan gran parte del espacio del que se dispone en los layouts, se decidió que podría ser una buena idea diseñar e incluir una serie de iconos que fueran fácilmente interpretables para que con únicamente un vistazo, los usuarios fueran capaces de usarlos entendiendo perfectamente cuál es el resultado que obtendrán al utilizarlos. Esto suponía que sólo era posible utilizarlos en aquellas áreas que debido a su funcionalidad fueran sencillos de representar gráficamente. Todos ellos se generaron tomando como base otros existentes y editándolos a través de la herramienta Adobe Photoshop 6 se adaptaron a los estilos que ya se estaban utilizando para dar un aspecto homogéneo y estable a la aplicación. Principalmente se modificaron las formas, los colores y los recuadros para darles forma de botón.

Finalmente, se crearon los siguientes iconos:

- Un aspa que se utilizaría para cerrar los mensajes de aviso, tanto los de error como los de éxito por lo que se podrán encontrar prácticamente en todas las secciones, ya que se utilizará para notificar al usuario de cambios
- Un lápiz de escritura que representaría la activación del modo edición. Este icono aparecerá únicamente en la sección '*Ver dashboards'*
- Un disquete de guardado que con el tiempo se ha convertido en el símbolo gráfico universal del almacenaje. En este caso sólo se incluirá en la sección de '*Ajustes*' lugar en el que se almacenará el endpoint de la aplicación
- Una papelera para mostrar todo tipo de borrados y eliminaciones, que en este caso se ha utilizado para borrar elementos almacenados en las preferencias compartidas, como los widgets, dashboard o la dirección del endpoint
- Por último, y como se ha visto en uno de los puntos anteriores se generó una flecha apuntando hacia abajo para representar en los controles spinner el despliegue de las opciones como por ejemplo el spinner del listado de nombres de dashboards o el listado de tipos de widgets que se puede ver en la sección de '*Añadir widget'*

Figure 55: Iconos de la aplicación

3.6.4. Actualización de versión Android

Para finalizar el desarrollo se revisó el archivo Gradle y al organizar las librerías se observó que durante desarrollo del proyecto, se había generado y desplegado una nueva versión de Android, la v25, por lo que se procedió a la instalación de la nueva versión y se sincronizó el proyecto con esta, de tal manera que la aplicación pueda ser también utilizada por los últimos dispositivos móviles que cuenten con esta API de Android, ya que como se ha llegado a comentar en secciones anteriores, en este desarrollo se pretendía soportar el máximo número o porcentaje de dispositivos siempre y cuando esto no repercutiera negativamente en el desarrollo o en el rendimiento del mismo, es importante recordar que este punto es esencial.

Además de esa actualización, se realizó lo propio con las librerías de soporte, ya que es importante que mantengan la misma versión Android base para librarse de incompatibilidades y cualquier otro tipo de imprevisto.

4. Medios empleados

En este capítulo se hablará de los medios, tanto hardware como software que se han utilizado para la realización de este proyecto.

4.1. Software

Para la parte software existían distintas herramientas que se utilizaban para desarrollar la aplicación. El proyecto Android fue generado a través de un entorno de desarrollo integrado (IDE a partir de este punto) llamado Android Studio.

Android Studio es un software creado por Google, cuya primera versión estable fue presentada al público en 2014. Este programa se creó además de para proporcionar una opción de IDE más focalizada en el desarrollo de aplicaciones Android, para desbancar a Eclipse como IDE preferido por los desarrolladores de aplicaciones móviles Android.

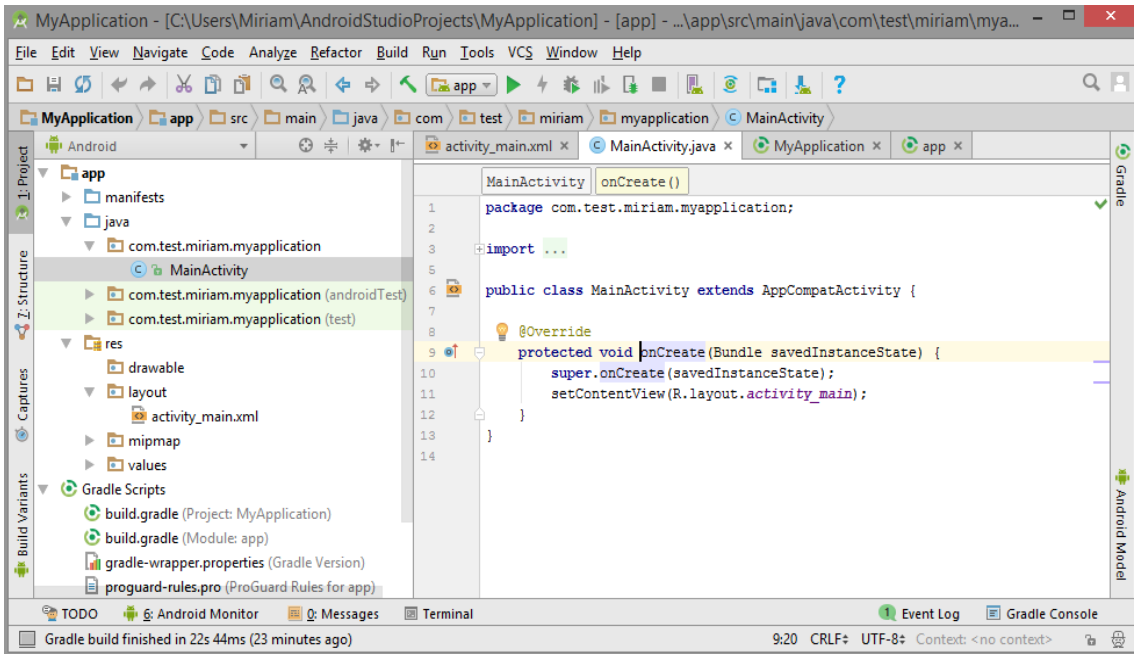

Figure 56: Interfaz AndroidStudio

Este software ofrece a los usuarios tanto la típica sección de edición de código con su explorador de archivos como una sección visual y de diseño específica para los layouts de las vistas del usuario. Esta sección de diseño simplifica mucho el trabajo de los desarrolladores porque permite una renderización básica de la vista sin tener que compilar todo el proyecto y generar una APK que se instale en el dispositivo. Es en otras palabras, una vista previa de los layouts con los que además es posible interactuar, ya que la interfaz gráfica proporciona distintos widgets y etiquetado de elementos directamente desde esta vista y los incorpora a continuación al código del layout con las mismas características que se añadieron con la interfaz de diseño. Lo mejor de todo, es que esta vista es capaz de actualizarse al momento y por lo tanto ofrece una visión perfecta del estado de las vistas o layouts del usuario.

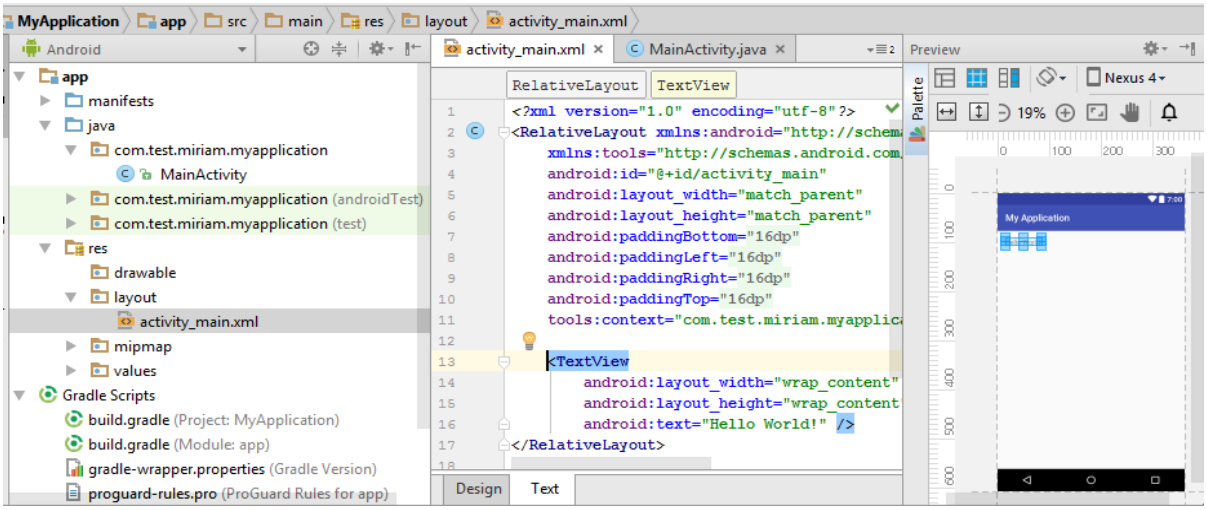

Figure 57: Interfaz Preview de AndroidStudio

Dentro de todo lo que ofrece Android Studio, probablemente la opción del uso de los emuladores sea la menos aprovechada durante el desarrollo y las pruebas del proyecto. No obstante, sí que se ha llegado a utilizarlos en alguna ocasión y de hecho, se han diseñado tres emuladores con características distintas para realizar pruebas concretas como el cambio de archivos de tipo "Values" y ver su funcionamiento en un entorno más cercano al real. La razón por la que no se ha llegado a utilizar tanto como cabría esperar, es porque requiere una gran cantidad de recursos del equipo y ralentizaba horriblemente cualquier tipo de acción que se quisiera probar, así que realmente la aplicación se ha limitado a pruebas muy simples usando los emuladores.

Una de las características a las que más uso se ha dado es al sistema de control de versiones que incluye el paquete de software. Android Studio permite conectar el proyecto a un repositorio para poder actualizar y realizar las subidas desde el mismo IDE. Actualmente tiene tanto un desarrollo para SVN como para GIT, que es el que se ha utilizado en este caso. No tener que depender de otro software para las subidas a la nube hace muy cómodo el trabajo del desarrollador y facilita la gestión de los cambios con el código de colores que el programa aplica a estos ficheros para que se detecte qué es nuevo y qué se ha modificado.

Otra de las características de Android Studio son sus actualizaciones, ya que el equipo detrás de su desarrollo no para de añadir nuevas funcionalidades y de corregir errores. De esta manera se consigue un IDE muy potente y que está al día de los avances. Las actualizaciones son automáticas y se notifican cada vez que se abre el programa, aunque el usuario puede decidir no llevarlas a cabo o postergarlas para un momento más propicio. Por supuesto, aunque no se actualice el programa sigue funcionando tal lo cual lo hacía antes de tener disponible la actualización, es decir, que no se bloquea y se deja al usuario la toma de decisiones. Por otro lado, las continuas actualizaciones pueden ser algo pesadas, ya que es bastante usual que salgan varias notificaciones al mes con cambios y no todas son de rápida aplicación.

Un detalle interesante del desarrollo Android que se ha comentado al principio de esta sección, es que hasta hace no mucho tiempo, era posible utilizar otros IDEs y existían varias alternativas a Android Studio, pero algunos de ellos ya no reciben soporte por parte de los desarrolladores. Entre ellos se encontraban tanto Xamarin como Eclipse. De hecho, Eclipse era la opción por defecto para muchos desarrolladores en los inicios de esta tecnología y es que hasta el momento se utiliza Java como lenguaje nativo, que junto a C++ están considerados los lenguajes oficiales para Android, y en este caso, aunque es un IDE que se ha utilizado, no se ha continuado con su uso por el hecho de que ya no recibe ningún tipo de soporte y puede considerarse que se encuentra obsoleto en la actualidad. Aun así y cómo se ha comentado, en un inicio era la opción por defecto porque muchos desarrolladores ya lo conocían, ya que se trata de uno de los IDEs más extendidos para Java y por lo tanto era más sencillo e intuitivo para estos usuarios. Además, cabe destacar que Eclipse tenía un funcionamiento

más fluido y requería menos recursos del equipo que los que necesitaba Android Studio, pero estaba bastante claro que no era un IDE creado específicamente para Android y fue quedándose por detrás, hasta que al final Android Studio acaparó la mayor parte del mercado y cuando decimos la mayoría del mercado no se está exagerando, ya que según algunas fuentes acapara casi el 80%.

4.2. Hardware

Para la parte hardware, se contaba primeramente con varios dispositivos móviles Android para realizar las pruebas de la aplicación, un portátil en el que se tenía instalado el software de Android Studio y por supuesto, un cable USB que al usuario servía para conectar el terminal con el equipo.

El terminal del que se hablaba en el párrafo anterior, es un móvil Android bq modelo Aquaris 5 HD, el cual tiene la versión 4.4.2 de Android, comúnmente llamado KitKat, y por lo tanto cuenta con la versión 19 de la API. El dispositivo tiene una resolución de 1280x720 píxeles, 5 pulgadas de pantalla, una densidad de píxeles de 294 y lo que es más importante, un procesador Cortex A7 de 4 núcleos con una memoria RAM de 1Gb.

También se ha usado un segundo terminal que era un móvil Samsung Galaxy modelo Trend. El terminal tenía la versión 4.0.2 de Android, comúnmente llamado IceCreamSandwich, y por lo tanto contaba con la versión 15 de la API.

Además de un dispositivo en el que instalar la APK resultante de este proyecto, era necesario utilizar un ordenador para realizar el desarrollo. En este caso se ha contado con un equipo portátil Toshiba Satellite con un procesador Intel Core i5 y una memoria RAM de 4 Gb. Además de necesitar una memoria RAM medianamente potente también se necesita un equipo con un disco duro que tenga al menos unos cuantos Gb libres de memoria, ya que por un lado únicamente el programa ocupa un buen puñado de Gb, pero si a esto se le incluye la instalación del JDK (Java development kit), del paquete de herramientas de Android SDK (Software development kit) y de las imágenes de sistema que se crean para los emuladores, se juntarán con una cantidad importante de memoria del disco duro, pudiendo alcanzar fácilmente los 30Gb, cuando según las especificaciones del programa se necesitan únicamente unos 2Gb, aunque recomiendan 4Gb. En cuanto a la memoria RAM, como se exponía, es necesaria una memoria bastante potente, se recomienda 8Gb, aunque puede ejecutarse fácilmente sobre 3Gb y si se utilizasen emuladores de dispositivos se tendría que añadir 1Gb más. Esto supone que no cualquier equipo es capaz de usar el software y que utilizar los emuladores causa bastantes ralentizaciones en las renderizaciones de las vistas y demás funcionalidades, ya que consume demasiados recursos. Debido a ello, no ha quedado más remedio que simular y realizar todas las pruebas directamente desde un terminal móvil en el cual se instalaba la APK que se compilaba a través del Gradle y de la SDK. De esta manera, se ganaba y aprovechaba tiempo, ya que era más fluido y cómodo de usar, y se probaba directamente sobre un entorno real.

5. Futuras mejoras

Una vez habiendo finalizado el desarrollo de la aplicación, se han estudiado e ideado nuevas funcionalidades que podrían formar parte de la misma y se decidió recogerlas en este documento y enumerarlas en un apartado independiente.

5.1. Seleccionar primario después de borrado

Actualmente y como se ha visto en el apartado 3.5.14. Modo eliminación, al borrar un dashboard, si este no era el primario, simplemente se producía la eliminación y se cargaba de nuevo la vista, pero esta vez con el dashboard principal a la cabeza. En otras palabras, al renderizarse de nuevo la vista "*Ver dashboard*" se preseleccionaba por defecto en el spinner el dashboard marcado como principal. En el caso de que el dashboard a eliminar fuera el primario o principal, una vez eliminado, se procedía a asignar otro dashboard como principal, siendo el elegido en este caso, el primero que pueda encontrarse en el listado. En caso de no quedar ningún dashboard en la lista, por supuesto no será posible asignar esa propiedad y aparecerá un mensaje de que no existe ningún dashboard guardado en el terminal.

La mejora que se aplicaría sobre esta cuestión sería que a la hora de borrar un dashboard que tuviese la propiedad de principal, en lugar de que esa propiedad se asignara por defecto al primer dashboard de la lista, permitiera elegir al usuario entre los restantes.

Esta mejora supondría mostrar ese listado de dashboards en un panel desde el cuál se iniciara la acción de asignar la propiedad al dashboard seleccionado en el listado. Por ello, se debería a priori recorrer el listado para encontrar el elegido por el usuario, modificar la propiedad y actualizar estos datos en el terminal.

5.2. Varios endpoint

En la aplicación actual únicamente se permite al usuario almacenar un endpoint al que poder acceder y del que se podrán obtener datos externos.

Lo que se vendría a proponer con esta mejora sería generar todo un sistema de creación y gestión de endpoints similar a lo que se ha desarrollado en la aplicación en estos momentos para los dashboards. Se debería permitir no sólo la creación, sino la edición y eliminación de los mismos, y claro, tampoco estaría de más tener la opción de priorizar alguno de ellos. Luego en los propios widgets al igual que se elige a qué dashboard van a pertenecer se podría seleccionar de qué endpoint tomarán los datos externos. Por supuesto, esto último solamente se realizaría para aquellos widgets que pueden recoger datos del exterior, o lo que es lo mismo el widget de línea temporal y el widget de dato individual.

Esta mejora permitiría al usuario poder contar con diversos proyectos IoT y conseguir estar al tanto de la información que reciben de ellos de manera visual. Con lo que cuenta ahora la aplicación, únicamente son capaces de monitorizar uno, así que poder permitir esa flexibilidad a la aplicación podría ser muy útil a medio y largo plazo.

5.3. Seleccionar los valores del endpoint

En esta aplicación, para los widgets de tipo línea temporal y dato individual, es el usuario el que debe conocer el nombre del campo al que quiere acceder. Información que será proporcionada por el endpoint en cuestión.

Esta mejora, lo que plantea es la posibilidad de leer previamente los campos que envía el endpoint en las muestras o registros y sacar un listado con ellos. De esta manera, el usuario sería capaz de visualizar los campos a los que puede acceder simplemente a través del layout de formulario de adición del widget.

Por supuesto, este campo sería únicamente visible para aquellos widgets que recojan sus valores de

un endpoint y tendría forma de spinner. Para ello se debería leer las entradas de los registros que proporciona el endpoint, y con ellos se precargaría el componente. Además, en el caso de que ya estuviese disponble la mejora 6.2 Varios endpoints, habría que controlar cuando cambia el endpoint seleccionado y recargar el spinner de esta mejora con los datos actualizados, o en otras palabras filtrar los valores de los nombres de los campos, basándose en el endpoint seleccionado por el ususario.

5.4. Mover widget de un dashboard a otro

Uno de los principales puntos clave que se han mantenido en la aplicación desde el inicio de su desarrollo hasta el final, es que un widget pertenece a un dashboard y únicamente a uno. Derivado de este punto, se podía extraer que si el usuario quería contar con un widget que ofreciera la misma información que otro, entonces el usuario debía crear un widget nuevo a imagen y semejanza del anterior, lo cual no ofrecía ningún problema, facilitaba el mantenimiento de la aplicación y mantenía la independencia de los dashboards entre sí.

El problema que se ha observado es que en ocasiones no es que el usuario quiera contar con el mismo widgets dos veces (cada uno en un dashboard distinto), sino que lo que pretendía era que un widget que se visualizaba en el dashboard 'a' ahora se visualizase en el 'b'. Para ello con el desarrollo actual, el usuario debería crear un nuevo widget, añadirlo al dashboard 'b' y eliminar el igual en el dashboard 'a'.

Por ello se ha pensado esta mejora que se encargaría de añadir en el modo edición la posibilidad de que con el formulario de edición de widgets se incorpore un nuevo campo de tipo Spinner en el que salga seleccionado el dashboard al que pertenece el widget, pero que dicho Spinner además de esa opción contenga el resto del listado de dashboards para poder escoger a cual se quiere traspasar. De esta manera el formulario de edición pasaría a tener exactamente el mismo contenido que los fragmentos de creación de widgets con la diferencia en este caso de que el formulario de edición aparecería precargado con los datos del propio widget.

Una vez se haya llegado a este punto la aplicación se encargaría de recibir los cambios recibidos del usuario y eliminaría el widget del primer dashboard para añadirlo al final de la lista de widgets del que haya sido elegido.

Con esas modificaciones, la aplicación daría la impresión de haber transferido el widget cuando en realidad lo que se han hecho son varias acciones de borrado y adición.

5.5. Ordenación de widgets

En estos momentos, en la aplicación, una vez que se adhiere un widget este se queda colocado todo el tiempo en la misma posición dentro del listado de widgets y su única variación se produce cuando se elimina cualquiera del resto de widgets que se encuentran por delante de él.

Lo que se propondría con esta mejora sería poder adelantar o atrasar el widget dentro del listado según los intereses del usuario y esto tendría un par de variantes de desarrollo:

- Por un lado se podría contar con dos iconos de flechas, uno que apuntase hacia arriba y otro que lo hiciera hacia abajo. Cada uno de ellos activaría una acción de desplazamiento de los widgets. Para ello, como desde el código no se pueden variar las posiciones de los elementos dentro de los listados, se obtendría la posición actual del widget, se le sumaría o restaría una unidad, según la flecha elegida, y se almacenaría en una variable el objeto por el completo. A partir de este punto, se continuaría eliminando dicho widget del listado y añadiendo el objeto almacenado en la variable que era una copia del eliminado. Como sí que es posible una adición de un elemento en una posición concreta, se utilizaría la posición que se guardó al principio para hacerlo. Por supuesto, si el widget fuera el primero, no se visualizaría la flecha que apunta hacia arriba, de igual manera que si fuera el último, no se pintaría en la vista la que apunta hacia abajo. Esto se haría para evitar acceder a posiciones no existentes (-1) y para no tener

huecos vacíos a la hora de iterar sobre el listado. Esta versión supone que para producirse varios desplazamientos se debe activar la opción tantas veces como posiciones se deseen escalar o descender, pero sería relativamente trivial en cuestión de tiempo, por lo que debe ser descartada

- Otra variante sería que en lugar de contar con flechas para desplazar los widget, se tendría un campo de edición de texto que únicamente aceptara números. En ese EditText el usuario podría escribir directamente la posición a la que quiere que se desplace el widget. Ese valor sería validado por la aplicación y si cumpliera las condiciones, se eliminaría la instancia del widget una vez este tuviera una copia, la cual se añadiría al listado parametrizando la posición con el valor del campo de edición
- Estas dos variantes anteriores podrían convivir perfectamente en la aplicación aunque para ello deberían trabajar de manera independiente
- Otra variante más que se ha investigado sería hacer que estos fragmentos aceptaran una funcionalidad de tipo drag&drop. De esta manera, el usuario mantendría pulsado un widget un determinado plazo de tiempo, momento en el cual se mostraría algún indicativo de que se encuentra en modo drag&drop y sin levantar el dedo de la pantalla podría desplazar el widget manualmente. Una vez que el usuario levantara el dedo, se deberían llevar una serie de acciones, entre las que estarían conseguir las posiciones de todos los widgets para modificar su orden en el listado

Cabe destacar que cada modificación de posiciones sería independiente y afectaría a un único widget, ya que el proceso consistiría en activar el cambio de posiciones, desplazar el elemento y volver a cargar la vista del fragmento ShowDashboard. Además, estas modificaciones únicamente se realizarían si el modo edición se encuentra activado y los resultados de los cambios de orden deberán ser almacenados en el terminal.

5.6. Imágenes internas y externas

En estos momentos y como se ha visto en el apartado de Diseño y desarrollo, existe en la aplicación un widget que permite la inclusión de imágenes para que se visualicen en el dashboard.

Dichas imágenes se encuentran almacenadas en el dispositivo móvil y la aplicación se encarga de encontrarlas en la memoria, tanto interna como externa, del terminal a través de su ruta de acceso que se obtiene desde la propia galería de imágenes de Android.

Lo que se pretende con esta mejora es que además de mostrar imágenes internas también permita cargar externas, directamente desde internet.

Para ello el funcionamiento de carga podría mantenerse tal cual está en estos momentos, ya que las funciones que utiliza de carga obtienen la imagen a través de una URI que puede ser tanto interna del teléfono como externa, de la red.

Donde variaría sería a la hora de obtener dicha URI, se necesitaría algún tipo de mecanismo que al variarlo se activara un layout distinto para cuando se quiere acceder a la galería de imágenes o cuando se quiere conseguir la URI de la red. Ese sistema podría crearse tanto con un Spinner al igual que se ha hecho en otras ocasiones o con un RadioButton con dos opciones que mostrara y ocultara los layouts por su cuenta.

En el caso de una URL el campo a pintar sería un EditText para que la URL externa se copie de internet y se pegue en el proyecto. Para el caso de la galería de imágenes se mantendría tal cual está ahora mismo.

5.7. Generación de versiones idiomáticas

Esta mejora que se propone en este apartado no ofrecería una funcionalidad nueva de la que el usuario pudiera disfrutar, pero sigue siendo un punto importante si se quiere que la aplicación amplíe al máximo su cuota de mercado.

Actualmente la aplicación mantiene todos sus textos en español (a excepción de palabras como dashboard o widget que para el proyecto son más intuitivas en inglés). Esto supone que existirá una cantidad importantísima de usuarios que no la utilizarán al no estar localizada en su idioma. Por esa misma razón se ha propuesto aumentar la oferta de idiomas disponibles y como en este caso todos los literales textuales se encuentran en el fichero strings.xml se gestionaría la mejora, añadiendo el del resto de idiomas a los que la aplicación se encuentre traducida.

6. Conclusiones

Para finalizar este documento es importante recalcar que los objetivos propuestos en la introducción han sido cumplidos al finalizar el desarrollo del proyecto.

Se ha conseguido crear una aplicación Android desde la que los usuarios podrían controlar y monitorizar los cambios en sus proyectos IoT de forma remota, ya que se reciben los datos a través de la red.

A pesar de todo ello, es importante tener en cuenta que leyendo las líneas futuras de trabajo, este proyecto tiene grandes posibilidades de crecer en el tiempo y que su escalabilidad es clave para poder ofrecer a los usuarios nuevas y mejores funcionalidades y características de cara a tener una aplicación que sirva para cubrir todas aquellas necesidades que un desarrollador de proyectos IoT pueda tener. Además, de ofrecer nuevas acciones sería interesante poder explotar al máximo las posibilidades que ofrece la librería 'HelloCharts', ya que es bastante más potente que lo que se ha podido demostrar durante el desarrollo y en este mismo documento.

Además de todo lo que ofrece el proyecto, creo que era importante elegir una plataforma que se adaptase a los potenciales usuarios de la misma y por ello, la elección de Android como tecnología de la aplicación, ha sido la adecuada, principalmente porque gran parte de los usuarios que están desarrollando proyectos IoT de manera individual e independiente usan o dispositivos Android o IOS, así que era importante cubrir uno de los dos grandes mercados actuales de dispositivos móviles.

Por último, este proyecto ha supuesto un aprendizaje importante en esta tecnología que tantas salidas y posibilidades ofrece en la actualidad. Además, era importante para finalizar el proyecto, que este aprendizaje sirviera en el futuro para poder continuar desarrollándose en esta línea de trabajo, ya sea de manera profesional o de manera independiente y que demostrara que después de años de estudio todo lo aprendido en la universidad ha servido para poder crear un proyecto de principio a fin en una tecnología desconocida hasta ese momento por mí.

7. Planificación

Para llevar un proyecto de este calibre y en realidad para la mayoría de proyectos, es esencial contar con una planificación y con alguna herramienta de monitorización para poder controlar los avances del mismo y ver si se es capaz de cumplir los plazos estipulados o por el contrario el desarrollador se ve en la necesidad de reestimar los valores dados en un inicio.

Lo primero que se debes hacer en la fase de planificación inicial es la de desglosar las tareas con las que se contarán en el proyecto:

- Fase de análisis → esta primera fase consistirá en definir cuáles son los requerimientos del proyecto y en ella se tendrán la mayor parte de las reuniones para eliminar todas las dudas relacionadas con él y para asegurar con qué funcionalidades debe contar una vez esté terminado. En esta parte, el estudio del arte es fundamental para poder definir cuál es el problema que se tiene entre manos

- Fase de diseño técnico → una vez que se es conocedor de qué problema se quiere solucionar con el proyecto, se deberá definir cómo se va a llevar a cabo. Por ejemplo, en un caso como este, en esta fase se definiría la tecnología a usar y se utilizaría las conclusiones de todas las investigaciones realizadas en el segmento anterior para tomar las decisiones sobre el desarrollo

- Fase de desarrollo y documentación → en este punto se realizaría la solución planteada en términos técnicos con la tecnología escogida para ello y paralelamente se llevaría a cabo la documentación de todos los pasos para que se pueda llevar un control del desarrollo

- Fase de pruebas y validación → una vez finalizado e incluso durante el desarrollo del mismo, se pasaría a realizar todo tipo de pruebas para detectar en una fase temprana los errores y poder corregirlos lo antes posible para que no afecten al uso de la aplicación por parte de los usuarios

- Fase de implantación → una vez dado el visto bueno al desarrollo ya se podría comenzar a implantarlo en los portales de descargas para que los usuarios tengan acceso a ello. En este caso al ser una aplicación móvil Android, se debería dar acceso a ella desde el playstore de la plataforma y que así los usuarios serían capaces de encontrarla e instalarla con la mayor facilidad posible

Metodología de trabajo

Para la planificación de estas fases puede realizarse siguiendo distintas metodologías de trabajo. Una de las más habituales es la del modelo en cascada que determina que no debe empezarse una fase hasta que la anterior hay concluido por completo, pero para este caso, un proyecto de este tipo se beneficiaría enormemente de un sistema de metodologías ágiles aunque no se haya podido implantar por completo debido a que no se contaba con todos los actores necesarios para ello, aun así, es lo indicado que algunas de las fases pueden solaparse como por ejemplo la de desarrollo y pruebas, e incluso la de diseño técnico, ya que en ocasiones los desarrolladores se han encontrado con desafíos técnicos que se han tenido tenido que solucionar, teniendo para ello que seguir pasos típicos de las primeras fases como la parte de investigación de las opciones técnicas y la implantación de la misma.

Todo lo que se ha comentado en esta sección sobre la metodología es muy importante, ya que como cabe suponer, a pesar de realizar una planificación inicial, ésta muy probablemente sufra cambios debido a los imprevistos que surgen durante estos procesos.

Herramienta de monitorización

Para controlar esas desviaciones en la planificación inicial se cuenta con una herramienta que permite conocer cuánto han variado las estimaciones del equipo y dónde ha sufrido más modificaciones este proyecto según valores de la planificación. De tal manera, que sea posible saber cuáles son las fases más proclives a sufrir retrasos o imprevistos.

La herramienta que se ha usado para ello en la mayoría de proyectos ha sido MSOffice Project. Este software está especialmente diseñado para el control y monitorización de proyectos, desde el desglose de tareas con sus respectivos tiempos de desarrollo hasta el control del presupuesto y la asignación de los recursos para cada uno de los registros tanto en términos de personal o de material y equipamiento. Con todos estos datos recogidos sobre un proyecto en concreto se puede seguir mucho más fácilmente el seguimiento y analizarlo.

Hasta hace poco era una de las herramientas más usadas por los jefes de proyectos, pero en los últimos 5 años y con la incorporación de las metodologías ágiles en los proyectos de las grandes corporaciones, se está pasando a nuevos modelos de control que mantengan el dinamismo de estas metodologías y que a eso se una el acceso online de todo el equipo, ya que deben participar activamente en el control y la planificación de las tareas que ellos mismos llevarán a cabo. Dentro de esas características se cuenta por ejemplo con las herramientas online de Trello o Jira, cada una de ellas con sus pros y sus contras, pero ambas válidas para ello.

En ellas se planifica en torno a tablones que representan los estados de las tareas, es decir, se contará con un tablón de ‘backlog’ o ‘to do’, ‘work in progress’, ‘done’ y ‘blocked’ para aquellas tareas que puedan encontrarse bloqueadas por otros desarrolladores o departamentos. Además las tareas se planificarán en términos de complejidad, lo que luego se traducirá en horas, pero siempre basándose en lo que cuesta realizar la tarea más básica y el número de subtareas que componen la tarea que se pretende planificar. Para este tipo de planificaciones y metodologías aunque puedan parecer algo caóticas de inicio, está totalmente demostrada su efectividad.

Task Name	Duration	Start	Finish	Prede
Análisis	**20 days**	**Mon 06/03/17**	**Fri 31/03/17**	
Análisis del problema	10 days	Mon 06/03/17	Fri 17/03/17	
Investigación de librerías	10 days	Mon 20/03/17	Fri 31/03/17	2
Diseño de la solución	**15 days**	**Mon 03/04/17**	**Fri 21/04/17**	
Definición de modelos de datos	5 days	Mon 03/04/17	Fri 07/04/17	3
Estructuración de la solución	10 days	Mon 10/04/17	Fri 21/04/17	5
Definición de módulos	8 days	Wed 12/04/17	Fri 21/04/17	5
Desarrollo	**37 days**	**Mon 24/04/17**	**Tue 13/06/17**	
Modelo de datos de la aplicación	1 day	Mon 24/04/17	Mon 24/04/17	5
Menú de navegación	4 days	Tue 25/04/17	Fri 28/04/17	6
Formularios de creación	5 days	Mon 01/05/17	Fri 05/05/17	9
Persistencia de datos	5 days	Mon 08/05/17	Fri 12/05/17	11
Fragmento 'Ver dashboard'	3 days	Mon 15/05/17	Wed 17/05/17	12
Widget reloj	2 days	Thu 18/05/17	Fri 19/05/17	13
Acceso a endpoints externos	5 days	Mon 22/05/17	Fri 26/05/17	14
Widget de dato individual	3 days	Mon 29/05/17	Wed 31/05/17	15
Widget línea temporal	4 days	Tue 30/05/17	Fri 02/06/17	15
Widget de imágenes	3 days	Mon 05/06/17	Wed 07/06/17	
Widget de mapas	4 days	Thu 08/06/17	Tue 13/06/17	
Pruebas	**40 days**	**Mon 08/05/17**	**Fri 30/06/17**	
Creación de dashboard	5 days	Mon 08/05/17	Fri 12/05/17	11
Creación de widgets	6 days	Fri 12/05/17	Fri 19/05/17	11
Visualización de widgets	20 days	Mon 22/05/17	Fri 16/06/17	14
Accesos endpoint externos	5 days	Mon 05/06/17	Fri 09/06/17	17;16
Flujo de edición	5 days	Mon 19/06/17	Fri 23/06/17	19
Flujo de eliminación	5 days	Mon 26/06/17	Fri 30/06/17	19
Documentación	15 days	Mon 03/07/17	Fri 21/07/17	26

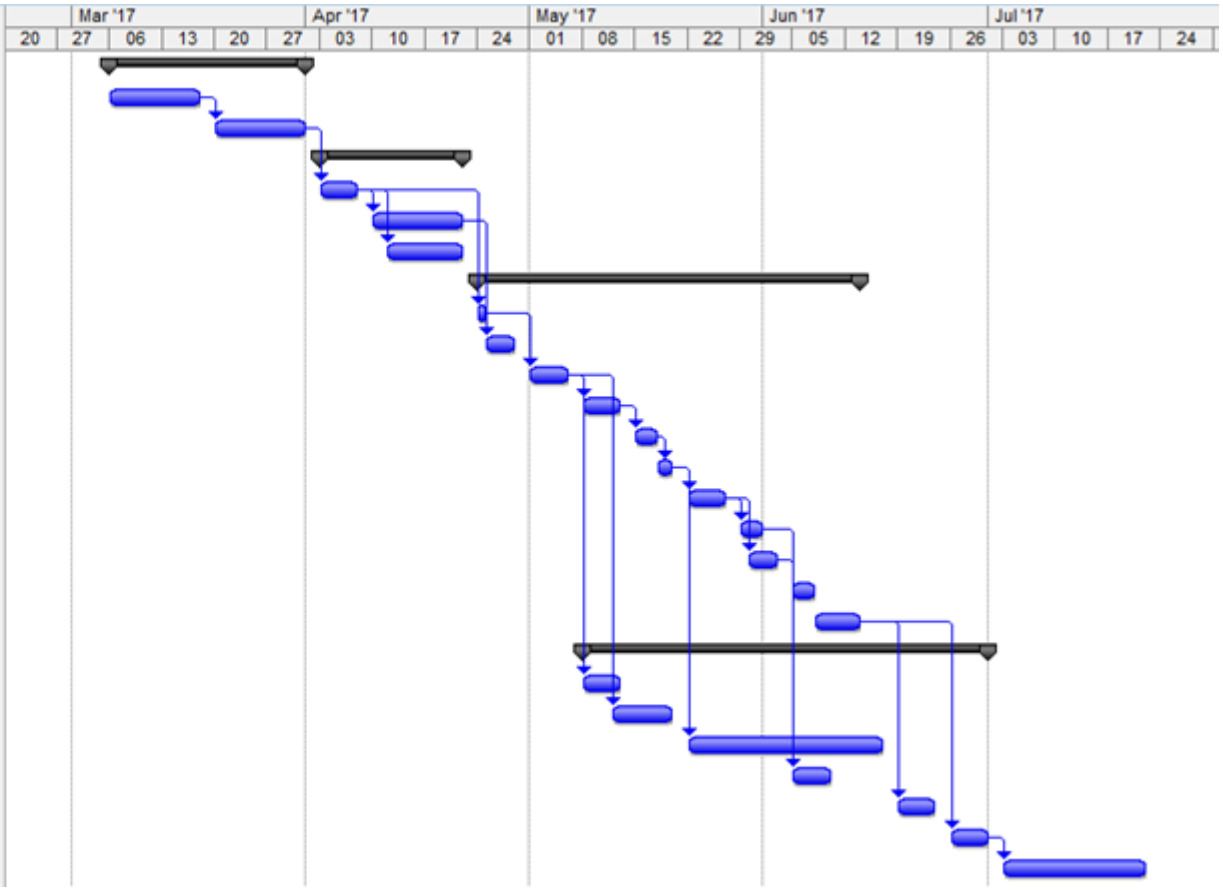

Figure 58: Diagrama Gantt planificación inicial

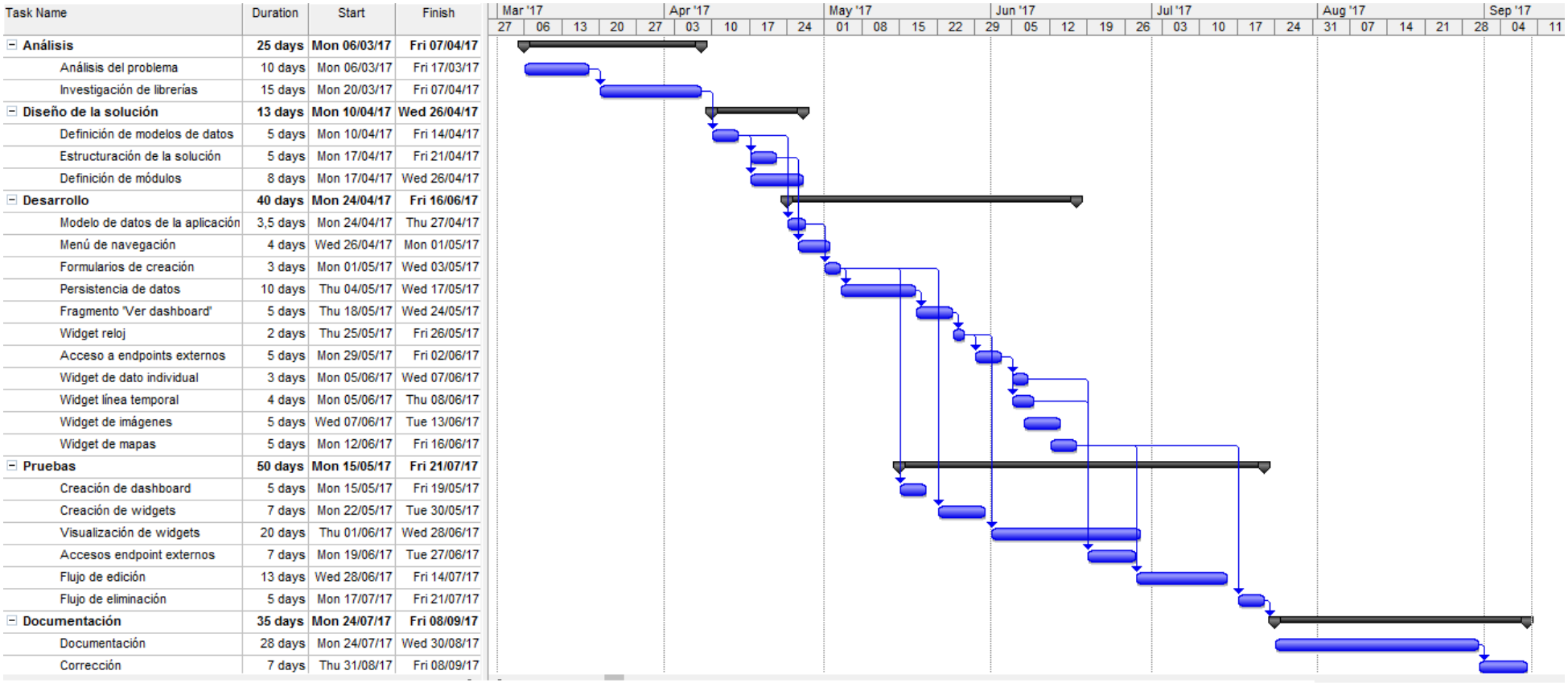

Figure 59: Diagrama Gantt planificación real

8. Presupuesto

En este apartado se detallará el presupuesto necesario para llevar a término un proyecto de estas características.

El presupuesto deberá definirse en base a los siguientes datos:

- Número de personas implicadas en el desarrollo
- Tiempo del que se dispone para realizar el desarrollo
- Material físico y energético que se utilizará
- Posibles compras de licencias o de diversos software

Aunque se intentará por parte del desarrollador ahorrar en costes hay que al menos tenerlo en cuenta y mantenerlo en todo momento presente por si fuera necesario aumentar el presupuesto en cualquiera de los puntos que se han visto.

Y bajo estos valores, será posible calcular cuál es el presupuesto final del proyecto. En este caso y en la mayoría de los proyectos que se rigen por estas mismas normas se contará con:

- Una única persona que se hará cargo del desarrollo
- Unos 6 meses de desarrollo, que podrá prorrogarse en caso de ser necesario, esto último dependerá no sólo de los imprevistos del proyecto sino también de las mejoras que puedan surgir en el desarrollo. Según las funcionalidades y requerimientos a llevar a cabo, el desempeño puede necesitar más tiempo
- Ordenador, ya sea de sobremesa o portátil con unas características mínimas que se estudió en el apartado de medios utilizados
- Terminales móviles en los que implantar el sistema desarrollado
- Dispositivos IoT, que en este caso son proporcionados de manera gratuita por el tutor del proyecto, aun así, se añadirá al presupuesto para que quede constancia de que esos gastos existen, aunque se hicieran con anterioridad
- En datos energéticos se enumerarán únicamente en los gatos de luz
- Compra de licencias MSSO y MSOffice

En la parte de compra de software se ha conseguido ahorrar bastante, ya que tanto Android Studio como las librerías que se han utilizado contaban con licencias de uso gratuito. Por lo que no se ha necesitado destinar más partidas presupuestarias a este tipo de gasto.

Personal				
Recurso	**Categoría**	**Coste/hora**	**Horas**	**Coste**
Miriam Calatrava	Desarrollador	€20,00	336h	€6720,00
			Total (€)	**€6720,00**

Tabla 1: Presupuesto personal

A partir de este punto se ha considerado incluir también el resto de elementos que se ha utilizado tanto en términos de hardware como de software aunque estos hayan sido de uso gratuito debido a que existe una distribución gratuita legal a la que se ha tenido acceso.

Según la ley del impuesto de sociedades de enero de 2015 se han calculado unos nuevos periodos de amortización para equipos electrónicos y software, así que se han apuntado en las tablas de este documento los valores legales en meses para sociedades y autónomos.

Software					
Recurso	**Coste/unidad**	**Unidades**	**% Uso proyecto**	**Periodo amortización**	**Costes**
Microsoft Windows 10 Pro	€279,00	1	100%	72	€23,25
Microsoft Office 2013	€124,32	1	50%	72	€5,18
Adobe Photoshop 6	€30,00	1	10%	72	€0,25
Android Studio	€0,00	1	100%		€0,00
HelloCharts	€0,00	1	25%		€0,00
Gson	€0,00	1	50%		€0,00
Repositorio Bitbucket	€0,00	1	75%		€0,00
			Total (€)		**€28,68**

Tabla 2: Presupuesto software

Hardware					
Recurso	**Coste/unidad**	**Unidades**	**% Uso proyecto**	**Periodo amortización**	**Costes**
Toshiba Satellite L650	€623,00	1	70%	120	€21,80
Toshiba Satellite Radius 12	€992,78	1	70%	120	€34,74
BQAquaris 5 HD	€120,00	1	80%	120	€4,80
Samsung Galaxy Trend	€89,00	1	15%	120	€0,66
Aparato IoT Arduino/Raspberry/etc.	€0,00	1	100%	120	€0,00
			Total (€)		**€62,00**

Tabla 3: Presupuesto hardware

Total	
Recurso	**Costes**
Personal	€6720,00
Software	€28,68
Hardware	€62,00
Total sin IVA	€6810,68
IVA (21%)	€1430,24
Total (€)	**€8240,92**

Tabla 4: Resumen de presupuestos

9. Referencias

[1] – S. Dieter Tebje Kelly, N. Kumar Suryadevara, and S. Chandra Mukhopadhyay, "Towards the Implementation of IoT for Environmental Condition Monitoring in Homes", 2013
http://ieeexplore.ieee.org/document/6516934/

[2] – R. Roman, P. Najera, and J. Lopez, "Securing the Internet of Things", 2011
http://ieeexplore.ieee.org/abstract/document/6017172/

[3] – J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions", 2013
http://www.sciencedirect.com/science/article/pii/S0167739X13000241

[4] – H. Ning and Z. Wang, "Future Internet of Things Architecture: Like Mankind Neural System or Social Organization Framework", 2011
http://ieeexplore.ieee.org/abstract/document/5722081/

[5] – M.U. Farooq, M. Waseem, S. Mazhar, A. Khairi, and T. Kamal, "A Review on Internet of Things (IoT)", 2015
https://www.researchgate.net/publication/273693976_A_Review_on_Internet_of_Things_IoT

[6] – A.D. Floarea, and V. Sgârciu, "Smart refrigerator: A next generation refrigerator connected to the IoT", 2017
http://ieeexplore.ieee.org/document/7861170/

[7] – D. Minoli, K. Sohraby, and B. Occhiogrosso, "IoT Considerations, Requirements, and Architectures for Smart Buildings—Energy Optimization and Next-Generation Building Management Systems", 2017
http://ieeexplore.ieee.org/document/7805265/

[8] – Li Da Xu, Senior Member, IEEE, Wu He, and Shancang Li, "Internet of Things in Industries: A Survey", 2014
http://ieeexplore.ieee.org/abstract/document/6714496/

[9] – John A. Stankovic, Life Fellow, IEEE, "Research Directions for the Internet of Things", 2014
http://ieeexplore.ieee.org/abstract/document/6774858/

[10] – M. Wolf, "The Physics of Event-Driven IoT Systems", 2016
http://ieeexplore.ieee.org/document/7748567/

[11] – BLYNK - Blynk, 2015, [Accedido Feb. 2017].
http://www.blynk.io/

[12] – CAYENNE - myDevices, 2016, [Accedido Feb. 2017].
https://mydevices.com/

[13] – A. Luis Bustamante, "Thinger", [Accedido Feb. 2017].
https://thinger.io/
https://github.com/thinger-io

[14] – Lecho, "HelloCharts", [Accedido Mar. 2017].
https://github.com/lecho/hellocharts-android

[15] – P. Jay, “MPAndroidChart”, [Accedido Mar. 2017].
https://github.com/PhilJay/MPAndroidChart

[16] – D. Nadeau, “Holo Graph Library”, [Accedido Mar. 2017].
https://bitbucket.org/danielnadeau/holographlibrary/wiki/Home

[17] – 4ViewSoft Company, “aChartEngine”, [Accedido Mar. 2017].
http://www.achartengine.org/content/demo.html

[18] – SciChart, [Accedido Mar. 2017].
http://www.scichart.com/android-chart-features

[19] – Syncfusion, “SyncfusionChart”, [Accedido Mar. 2017].
https://www.syncfusion.com/products/android/Chart

[20] – J. Chastang, “Charts4j”, [Accedido Mar. 2017].
https://github.com/julienchastang/charts4j

[21] – AndroidPlot, [Accedido Mar. 2017].
https://bitbucket.org/androidplot/androidplot/src

[22] – D. Bernardino, “WilliamChart”, [Accedido Mar. 2017].
https://github.com/diogobernardino/WilliamChart

[23] – S. Margaritelli, “Quartz”, “The easy way your “smart” coffee machine could get hacked and ruin your life”, 2017
https://qz.com/901823/the-easy-way-your-smart-coffee-machine-could-get-hacked-and-ruin-your-life/

[24] – Android Developers, “Dashboards”, 2017
https://developer.android.com/about/dashboards/index.html

[25] – Android Developers, “Activity”, 2017
https://developer.android.com/reference/android/app/Activity.html

[26] – Android Developers, “Storage Options”, 2017
https://developer.android.com/guide/topics/data/data-storage.html

[27] – C. Sharma, “Correcting the IoT History”, 2016
http://www.chetansharma.com/correcting-the-iot-history/

[28] – Adam D. Thierer, “SSRN Electronic Journal”, “Privacy and Security Implications of the Internet of Things”, 2013
https://www.ftc.gov/system/files/documents/reports/federal-trade-commission-staff-report-november-2013-workshop-entitled-internet-things-privacy/150127iotrpt.pdf

[29] – S. Kashyap, 10 Real World Applications of Internet of Things (IoT), 2016
https://www.analyticsvidhya.com/blog/2016/08/10-youtube-videos-explaining-the-real-world-applications-of-internet-of-things-iot/